お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

M16C/60 シリーズ ,M16C/20 シリーズ プログラム作成の手引き < アセンブリ言語編 >

はじめに

「M16C/60 シリーズ、M16C/20 シリーズ　アセンブリ言語プログラミングマニュアル」は、ルネサス CMOS16 ビットマイクロコンピュータ M16C/60 シリーズ、M16C/20 シリーズ用のアプリケーションプログラム開発において必要となる、基本的な知識を説明しています。本書のプログラミング言語はアセンブリ言語です。

M16C/60 シリーズ、M16C/20 シリーズを初めてお使いになる方は「第 1 章　M16C/60 シリーズ、M16C/20 シリーズの概要」を、CPU のアーキテクチャと命令については知りたい方は「第 2 章　CPU プログラミングモデル」を、アセンブラ指示命令について知りたい方は「第 3 章　アセンブラの機能」を、実践的なテクニックを知りたい方は「第 4 章　プログラミングスタイル」をご参照ください。M16C/60 シリーズ、M16C/20 シリーズの命令体系については、「M16C/60 シリーズ、M16C/20 シリーズ　ソフトウェアマニュアル」でさらに詳しく説明していますので、合わせてご利用ください。

なお、M16C/60 シリーズ、M16C/20 シリーズ各品種のハードウェアにつきましてはご使用品種のユーザーズマニュアルを、開発サポートツールにつきましては各ツールの操作説明書をご利用ください。

本書の使い方

本書は M16C/60 シリーズ、M16C/20 シリーズのアセンブリ言語プログラミングマニュアルです。M16C/60 シリーズの CPU コアをもつ全品種で共通に使用することができます。

本書を使用するにあたって、電気回路、論理回路、およびマイクロコンピュータの基本的な知識が必要です。

本書は 4 つの章から構成されています。以下に目的に応じた参照先（章、節）を示します。

○ M16C/60 シリーズ、M16C/20 シリーズの概要、特長を理解したい

→ 第 1 章　M16C/60 シリーズ、M16C/20 シリーズの概要

○アドレス空間、レジスタ構成、アドレッシングなどプログラミングに必要な事項を理解したい

→第 2 章　CPU プログラミングモデル

○ 命令の機能、書き方、使用できるアドレッシングを確認したい

→ 第 2 章　CPU プログラミングモデル　2.6　命令セット

○ 割り込みの使い方を知りたい

→ 第 2 章　CPU プログラミングモデル　2.7　割り込み

→ 第 4 章　プログラミングスタイル　　4.3　割り込み

○ 指示命令の機能、使い方を確認したい

→ 第 3 章　アセンブラの機能　3.2　ソースプログラムの書き方

○ M16C/60 シリーズ、M16C/20 シリーズのプログラミングテクニックを知りたい

→ 第 4 章　プログラミングスタイル

○ M16C/60 シリーズ、M16C/20 シリーズの開発手順を知りたい

→ 第 4 章　プログラミングスタイル　4.7　オブジェクトファイルの生成

目次

図目次

表目次

第 1 章

M16C/60 シリーズ、M16C/20 シリーズの概要

1.1 M16C/60 シリーズ、M16C/20 シリーズの特長

M16C/60 シリーズ、M16C/20 シリーズは組み込み機器を対象として開発されたシングルチップマイクロコンピュータです。

この節では M16C/60 シリーズ、M16C/20 シリーズの特長を紹介します。

M16C/60 シリーズ、M16C/20 シリーズの特長

M16C/60 シリーズ、M16C/20 シリーズは、使用頻度の高い命令を 1 バイトオペコードに配置しています。そのため、より小さなメモリ容量で効率の良いプログラムを書くことができます。

さらに、M16C/60 シリーズ、M16C/20 シリーズは 16 ビットマイコンでありながら 1 ビット /4 ビット /8 ビット処理を効率よく行います。1 クロックで実行できる命令も多数用意されています。そのため、高速な処理プログラムを書くことができます。

M16C/60 シリーズ、M16C/20 シリーズはリニアな 1M バイトのアドレス空間を持っています。したがって、プログラムサイズが大きくなる用途にも対応できます。

M16C/60 シリーズ、M16C/20 シリーズの特長を以下にまとめます。

(1)少ないメモリ容量で効率の良いプログラムを実現できる

(2)高速処理プログラムを実現できる

(3)1M バイトのアドレス空間を持っている

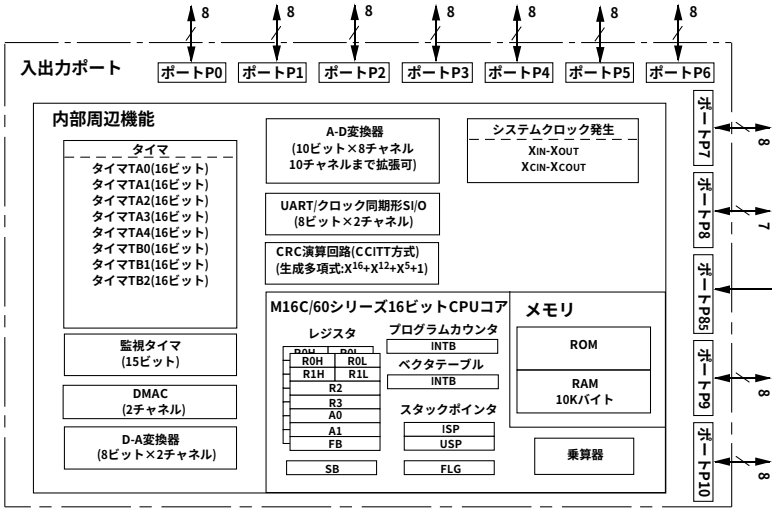
1.2 M16C/60 グループ、M16C/20 グループの概要

この節では M16C/60 シリーズの内部構成の例として M16C/60 グループを、M16C/20 シリーズの内部構成の例として M16C/20 グループを紹介します。M16C/60 グループは M16C/60 シリーズの、M16C/20 グループは M16C/20 シリーズのベースとなる製品です。詳細については、各品種のデータシート、ユーザーズマニュアルをご参照ください。

内部ブロック図

図 1.2.1 に M16C/60 グループ、M16C/20 グループのブロック図を示します。

(1)M16C/60 グループの場合



注 1.　M16C/61 グループの場合、UART/ クロック同期形 SI/O が +1 されます。
注 2.　M16C/62 グループの場合、UART/ クロック同期形 SI/O が +1、クロック同期形 SI/O が +1、タイマ B が +3 されます。

(2)M16C/20 グループの場合

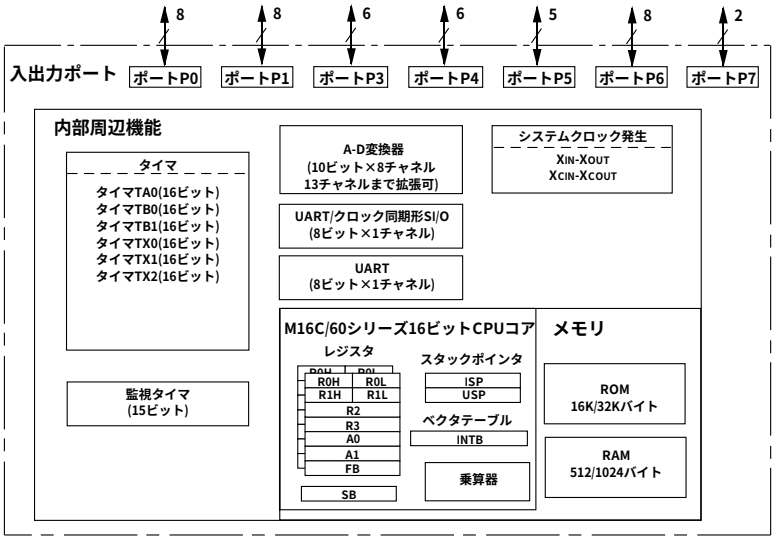


図 1.2.1　M16C/60 グループ、M16C/60 グループの概略仕様

M16C/60 グループの概略仕様

表 1.2.1 に M16C/60 グループの概略仕様を示します。

表 1.2.1　M16C/60 グループの概略仕様

項　目	内　容
電源電圧	2.7 ～ 5.5V（外部発振7MHz,1ウエイト時）
パッケージ	100ピン プラスチックモールドフラットQFP
動作周波数	10MHz (外部発振10MHz時)
最短命令実行時間	100ns (外部発振10MHz時)
基本バスサイクル	内部メモリ：100ns (外部発振10MHz時) 外部メモリ：100ns (外部発振10MHz, ノーウエイト時)
内部メモリ	ROM容量 / RAM容量 64Kバイト(注1) / 10Kバイト
動作モード	シングルチップモード, メモリ拡張モード, マイクロプロセッサモード
外部アドレス空間	1Mバイト (リニア) / 64Kバイト アドレスバス：20ビット / 16ビット
外部データバス幅	8ビット /16ビット
バス仕様	セパレートバス/マルチプレクスバス (チップセレクト信号4本内蔵)
クロック発生回路	2回路内蔵 (セラミック共振子または水晶発振子外付け)
内蔵周辺機能	
割り込み	内部17要因, 外部5要因, ソフトウェア 4 要因, 7レベル (キー入力割り込みを含む)
多機能１６ビットタイマ	タイマA 5本 + タイマB 3本
シリアル I/O	2本 (非同期 / 同期 切り替え可能)
A-D変換器	10ビット, 8 + 2 チャネル入力 (10 / 8ビット切り替え可能)
D-A変換器	8ビット, 2 チャネル出力
DMAC	2チャネル, 15要因
CRC演算回路	1回路内蔵
監視タイマ	15ビットカウンタ
プログラマブル入出力	87 本
入力ポート	1本 ($P8_5$、$\overline{NMI}$端子と兼用)

(注1) ROM外付け版M30600SFPを除きます。

M16C/20 グループの概略仕様

表 1.2.2 に M16C/20 グループの概略仕様を示します。

表 1.2.2　M16C/20 グループの概略仕様

項　目	内　容		
電源電圧	2.7 ～ 5.5V（外部発振7MHz,1ウエイト時）		
パッケージ	52ピン プラスチックモールドSDIP 56ピン プラスチックモールドQFP		
動作周波数	10MHz (外部発振10MHz時)		
最短命令実行時間	100ns (外部発振10MHz時)		
基本バスサイクル	内部メモリ：100ns (外部発振10MHz時)		
内部メモリ	ROM容量	RAM容量	
	32 K バイト	1024バイト	
動作モード	シングルチップモード		
クロック発生回路	2回路内蔵 (セラミック共振子または水晶発振子外付け)		
内蔵周辺機能			
割り込み	内部9要因, 外部3要因, ソフトウェア 4 要因, 7レベル (キー入力割り込みを含む)		
多機能１６ビットタイマ	タイマA 1本 + タイマB 2本 + タイマX 3本		
シリアル I/O	2本 (1本は非同期形専用、1本は非同期 / 同期 切り替え可能)		
A-D変換器	10ビット, 8 + 5 チャネル入力 (10 / 8ビット切り替え可能)		
監視タイマ	15ビットカウンタ		
プログラマブル入出力	43 本		

1.3 CPU アーキテクチャの紹介

この節では M16C/60 シリーズ、M16C/20 シリーズの CPU アーキテクチャを紹介します。これらの内容は本書の第 2 章で各項目ごとに詳しく説明しますのでご参照ください。

レジスタ構成

レジスタ構成を表 1.3.1 に示します。レジスタ R0、R1、R2、R3、A0、A1、FB の 7 本は、2 セット用意されています。これらはレジスタバンク指定フラグによって切り替わります。

表 1.3.1 M16C/60 シリーズ、M16C/20 シリーズのレジスタ構成

項目	内　容		
レジスタ構成			
データレジスタ	16ビット × 4 R0 R1 R2 R3	(32ビット × 2) R2R0　R2　R0 R3R1　R3　R1	(8ビット × 4) R0　R0H　R0L R1　R1H　R1L
アドレスレジスタ	16ビット × 2 A0 A1	(32ビット × 1) A1A0　A1　A0	
ベースレジスタ	16ビット × 2 SB FB		
専用レジスタ	20ビット × 2 PC INTB 16ビット × 3 USP ISP FLG	(FLGの詳細)　b15 IPL b0　U I O B S Z D C　: (PC) IPL : ﾌﾟﾛｾｯｻ割り込み優先レベル (レベル0～7, 番号が大きい程優先度は高い。) (PC) : 割り込み発生時にPCの上位4ビットを退避。 U : スタックポインタ指定 (U = 0 のときISP, 1のときUSP使用) I : 割り込み 許可 (I = 1のとき許可) O : オーバフロー (オーバフロー発生時O = 1) B : レジスタバンク指定 (B = 0のときﾚｼﾞｽﾀﾊﾞﾝｸ0, 1のときﾚｼﾞｽﾀﾊﾞﾝｸ1使用) S : サイン (演算結果が負のとき S=1, 正のときS=0) Z : ゼロ (演算結果がゼロのときZ = 1) D : デバッグ (D = 1にするとシングルステップ動作を行う) C : キャリーまたはボロー	

アドレッシングモード

アドレッシングモードには 3 つのタイプがあります。
(1) **一般命令アドレッシング**……64K バイト(00000H ～ 0FFFFH)の領域をアクセスする。
(2) **特定命令アドレッシング**……1M バイト (00000H ～ FFFFFH)全領域をアクセスする。
(3)**ビット命令アドレッシング**…64K バイト(00000H ～ 0FFFFH)の領域をビット単位でアクセスする。
各タイプで使用できるアドレッシングを表 1.3.2 に示します。

表 1.3.2　M16C/60 シリーズ、M16C/20 シリーズのアドレッシングモード

項目	内　容		
アドレッシングモード	一般命令	特定命令	ビット命令
即値	○ imm : 8/16ビット	×	×
レジスタ直接	○ データレジスタ, アドレスレジスタのみ	○ R2R0 or R3R1 or A1A0 ※SHL, SHA, JMPI, JSRI命令のみ	○ R0 , R1 , R2 , R3 , A0 , A1のみ
絶対	○ abs : 16ビット(0～FFFFH)	○ abs : 20ビット(0～FFFFFH) ※LDE, STE, JMP, JSR命令のみ	○ bit,base : 16ビット(0～1FFFH)
アドレスレジスタ間接	○ [A0] or [A1] dspなし	○ [A1A0] dspなし ※LDE, STE命令のみ	○ [A0] or [A1] dspなし (0～1FFFH)
アドレスレジスタ相対	○ [A0] or [A1] dsp : 8/16ビット	○ [A0] dsp : 20ビットのみ ※LDE, STE, JMPI, JSRI 命令のみ	○ [A0] or [A1] dsp : 8/16ビット
SB相対とFB相対	○ [SB] dsp : 8/16bit (0～255 / 0～65534) ○ [FB] dsp : 8bit(-128～+127)	×	○ [SB] dsp : 8/11/16bit (0～31 / 0～255 / 0～8191) ○ [FB] dsp : 8bit(-16～+15)
スタックポインタ相対	○ [SP] dsp : 8ビット(-128～+127) ※MOV命令のみ	×	×
プログラムカウンタ相対	×	○ label .S : +2 ～ +9 : .B : -128 ～ +127 .W : -32768 ～ +32767 ※JMP, JSR命令のみ	×
専用レジスタ直接	×	○ INTBL, INTBH, ISP, USP, SB, FB, FLG ※LDC, STC, PUSHC, POPC命令のみ	×
ＦＬＧ直接	×	×	○ U, I, O, B, S, Z, D, Cﾌﾗｸﾞ ※FCLR, FSET命令のみ

命令セット

表 1.3.3 に M16C/60 シリーズ、M16C/20 シリーズの命令を機能別に示します。命令は全部で 91 種類あります。

表 1.3.3　M16C/60 シリーズ、M16C/20 シリーズの命令セット

項目	内　容	
命令セット	8ビット可変長：91命令	
データ転送命令 14命令	・転送命令	MOV, MOVA
	・プッシュ / ポップ命令	PUSH, PUSHM, PUSHA / POP, POPM
	・拡張データ領域転送命令	LDE, STE
	・4ビット転送命令	MOVDir
	・レジスタとレジスタ / メモリとの交換命令	XCHG
	・条件ストア命令	STZ, STNZ, STZX
演算命令 31命令	・加算命令	ADD, ADC, ADCF
	・減算命令	SUB, SBB
	・乗算命令	MUL, MULU
	・除算命令	DIV, DIVU, DIVX
	・10進加算命令	DADD, DADC
	・10進減算命令	DSUB, DSBB
	・インクリメント / デクリメント命令	INC / DEC
	・積和演算命令	RMPA
	・比較命令	CMP
	・その他(絶対値, 2の補数,符号拡張)	ABS, NEG, EXTS
	・論理演算命令	AND, OR, XOR, NOT
	・テスト命令	TST
	・シフト / ローテート命令	SHL, SHA / ROT, RORC, ROLC
分岐命令 10命令	・無条件分岐命令	JMP
	・条件付き分岐命令	JCnd
	・間接ジャンプ命令	JMPI
	・スペシャルページ分岐命令	JMPS
	・サブルーチンコール命令	JSR
	・間接サブルーチンコール命令	JSRI
	・スペシャルページサブルーチンコール命令	JSRS
	・サブルーチン復帰命令	RTS
	・加算(減算)＆条件分岐命令	ADJNZ, SBJNZ
ビット操作命令 14命令		BCLR, BSET, BNOT, BTST, BNTST, BAND, BNAND, BOR, BNOR, BXOR, BNXOR, BMCnd, BTSTS, BTSTC
ストリング命令　3命令		SMOVF, SMOVB, SSTR
その他の命令 19命令	・専用レジスタ操作命令	LDC, STC, LDINTB, LDIPL, PUSHC, POPC
	・フラグレジスタ操作命令	FSET, FCLR
	・OSサポート命令	LDCTX, STCTX
	・高級言語サポート命令	ENTER, EXITD
	・デバッグ装置サポート命令	BRK
	・割り込み関連命令	REIT, INT, INTO, UND
	・外部割り込み待ち命令	WAIT
	・ノー オペレーション命令	NOP

第 2 章

CPU プログラミングモデル

2.7 割り込みの概要

2.1　アドレス空間

M16C/60 シリーズ、M16C/20 シリーズのアドレス空間は 00000H 番地から FFFFFH 番地までの 1M バイトです。ここでは M16C/60 グループのアドレス空間とメモリ配置、SFR 領域、固定ベクタ領域について説明します。

2.1.1 動作モードとメモリ配置

M16C/60 グループは、シングルチップモード、メモリ拡張モード、マイクロプロセッサモードの 3 つのモードから 1 つの動作モードを選択します。動作モードによってアドレス空間と使用できる領域とメモリ配置が異なります。

アドレス空間

M16C/60 グループのアドレス空間を図 2.1.1 に示します。

00000H 番地～ 003FFH 番地は SFR（スペシャルファンクションレジスタ）領域です。　品種展開では 003FFH 番地から番地の小さい方向に SFR 領域を拡張します。

00400H 番地以降はメモリ領域です。品種展開では 00400H 番地から番地の大きい方向に RAM 領域を、FFFFFH 番地から番地の小さい方向に ROM 領域を拡張します。ただし、FFE00H 番地～ FFFFFH 番地は固定ベクタ領域です。

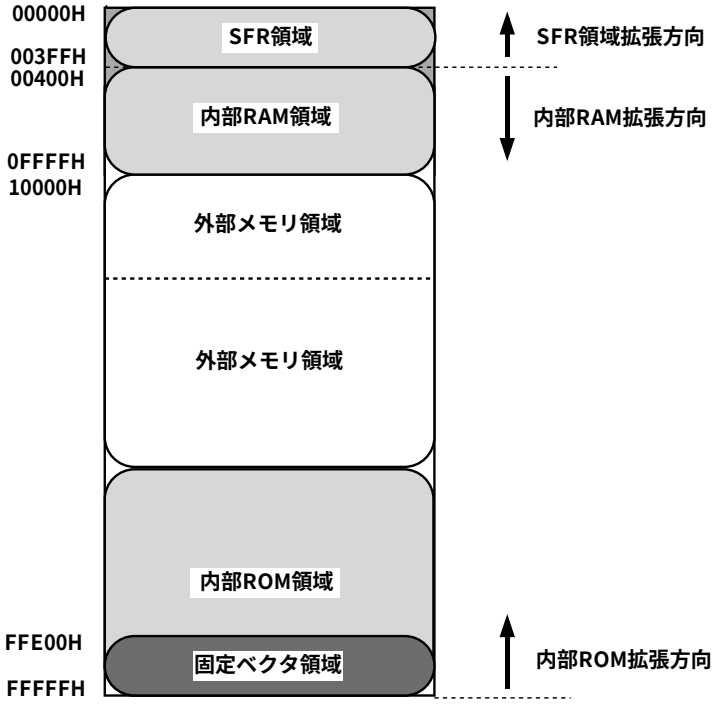


図 2.1.1　アドレス空間

動作モードとメモリ配置

●シングルチップモード

シングルチップモードは、内部領域（SFR、内部 RAM、内部 ROM）だけのアクセスが可能なモードです。

●メモリ拡張モード

メモリ拡張モードは、内部領域（SFR、内部 RAM、内部 ROM）および外部メモリ領域のアクセスが可能なモードです。

●マイクロプロセッサモード

マイクロプロセッサモードは、SFR および内部 RAM 領域と外部メモリ領域のアクセスが可能なモードです（内部 ROM 領域はアクセスできません）。

各動作モードのメモリ配置を図 2.1.2 に示します。

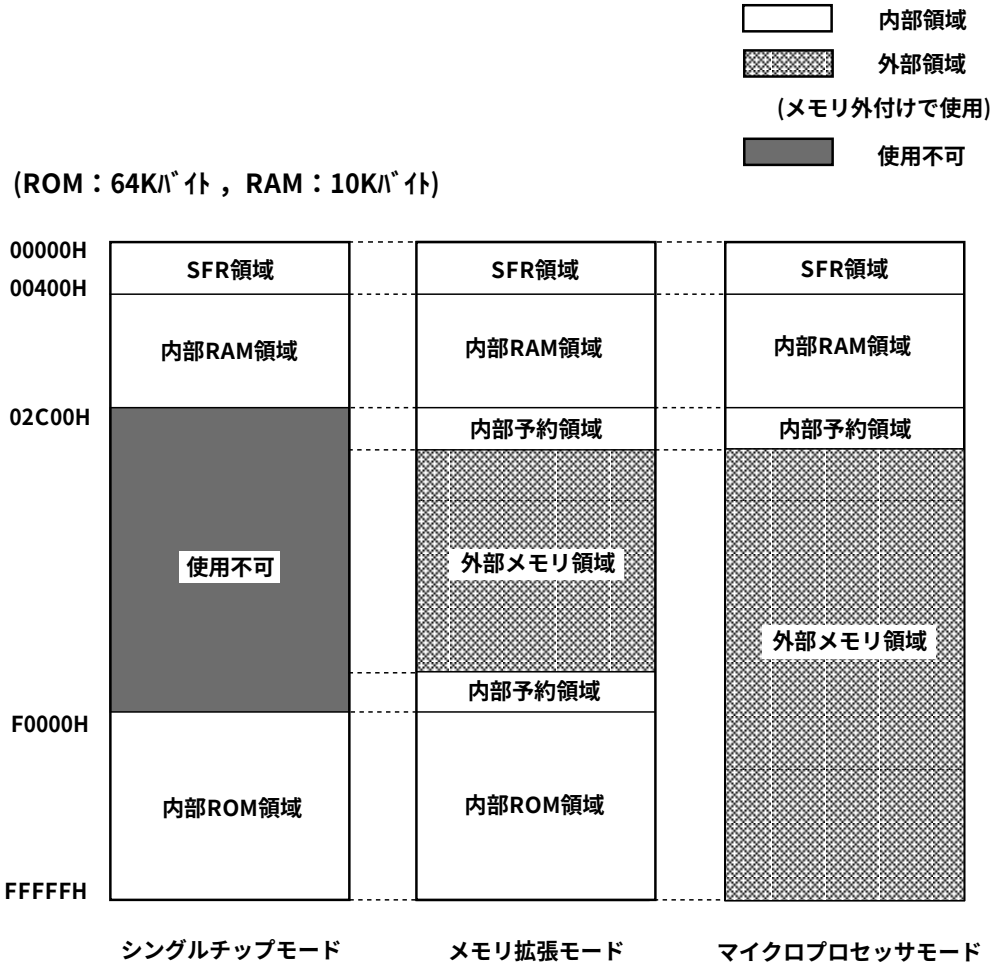


図 2.1.2　動作モー 2.1.2　SFR 領域

2.1.2 SFR 領域

SFR 領域には、動作モードを決定するプロセッサモードレジスタや、入出力ポート、A-D 変換器、UART、タイマなど周辺装置の制御レジスタが割り付けられています。制御レジスタのビット構成については M16C/60 グループのデータシートおよびユーザーズマニュアルをご参照ください。

なお、SFR 領域内の未使用領域は予約領域となっているため使用できません。

SFR 領域　制御レジスタの配置

SFR 領域　制御レジスタの配置を図 2.1.3 と 2.1.4 に示します。

0000₁₆	
0001₁₆	
0002₁₆	
0003₁₆	
0004₁₆	プロセッサモードレジスタ0(PM0)
0005₁₆	プロセッサモードレジスタ1(PM1)
0006₁₆	システムクロック制御レジスタ０(CM0)
0007₁₆	システムクロック制御レジスタ１(CM1)
0008₁₆	チップセレクト制御レジスタ(CSR)
0009₁₆	アドレス一致割り込み許可レジスタ(AIER)
000A₁₆	プロテクトレジスタ(PRCR)
000B₁₆	
000C₁₆	
000D₁₆	
000E₁₆	監視タイマスタートレジスタ(WDTS)
000F₁₆	監視タイマ制御レジスタ(WDC)
0010₁₆	アドレス一致割り込みレジスタ０(RMAD0)
0011₁₆	
0012₁₆	
0013₁₆	
0014₁₆	アドレス一致割り込みレジスタ１(RMAD1)
0015₁₆	
0016₁₆	
0017₁₆	
0018₁₆	
0019₁₆	
001A₁₆	
001B₁₆	
001C₁₆	
001D₁₆	
001E₁₆	
001F₁₆	
0020₁₆	DMA0ソースポインタ(SAR0)
0021₁₆	
0022₁₆	
0023₁₆	
0024₁₆	DMA0ディスティネーションポインタ(DAR0)
0025₁₆	
0026₁₆	
0027₁₆	
0028₁₆	DMA0転送カウンタ(TCR0)
0029₁₆	
002A₁₆	
002B₁₆	
002C₁₆	DMA0制御レジスタ(DM0CON)
002D₁₆	
002E₁₆	
002F₁₆	
0030₁₆	DMA1ソースポインタ(SAR1)
0031₁₆	
0032₁₆	
0033₁₆	
0034₁₆	DMA1ディスティネーションポインタ(DAR1)
0035₁₆	
0036₁₆	
0037₁₆	
0038₁₆	DMA1転送カウンタ(TCR1)
0039₁₆	
003A₁₆	
003B₁₆	
003C₁₆	DMA1制御レジスタ(DM1CON)
003D₁₆	
003E₁₆	
003F₁₆	

0040₁₆	
0041₁₆	
0042₁₆	
0043₁₆	
0044₁₆	
0045₁₆	
0046₁₆	
0047₁₆	
0048₁₆	
0049₁₆	
004A₁₆	
004B₁₆	DMA0割り込み制御レジスタ (DM0IC)
004C₁₆	DMA1割り込み制御レジスタ(DM1IC)
004D₁₆	キー入力割り込み制御レジスタ(KUPIC)
004E₁₆	A-D変換割り込み制御レジスタ(ADIC)
004F₁₆	
0050₁₆	
0051₁₆	UART0送信割り込み制御レジスタ(S0TIC)
0052₁₆	UART0受信割り込み制御レジスタ(S0RIC)
0053₁₆	UART1送信割り込み制御レジスタ(S1TIC)
0054₁₆	UART1受信割り込み制御レジスタ(S1RIC)
0055₁₆	タイマA0割り込み制御レジスタ(TA0IC)
0056₁₆	タイマA1割り込み制御レジスタ(TA1IC)
0057₁₆	タイマA2割り込み制御レジスタ(TA2IC)
0058₁₆	タイマA3割り込み制御レジスタ(TA3IC)
0059₁₆	タイマA4割り込み制御レジスタ(TA4IC)
005A₁₆	タイマB0割り込み制御レジスタ(TB0IC)
005B₁₆	タイマB1割り込み制御レジスタ(TB1IC)
005C₁₆	タイマB2割り込み制御レジスタ(TB2IC)
005D₁₆	INT0割り込み制御レジスタ(INT0IC)
005E₁₆	INT1割り込み制御レジスタ(INT1IC)
005F₁₆	INT2割り込み制御レジスタ(INT2IC)

図 2.1.3　制御レジスタの配置 1

0380₁₆	カウント開始フラグ(TABSR)
0381₁₆	時計用プリスケーラリセットフラグ(CPSRF)
0382₁₆	ワンショット開始フラグ(ONSF)
0383₁₆	トリガ選択レジスタ(TRGSR)
0384₁₆	アップダウンフラグ(UDF)
0385₁₆	
0386₁₆	タイマA0(TA0)
0387₁₆	
0388₁₆	タイマA1(TA1)
0389₁₆	
038A₁₆	タイマA2(TA2)
038B₁₆	
038C₁₆	タイマA3(TA3)
038D₁₆	
038E₁₆	タイマA4(TA4)
038F₁₆	
0390₁₆	タイマB0(TB0)
0391₁₆	
0392₁₆	タイマB1(TB1)
0393₁₆	
0394₁₆	タイマB2(TB2)
0395₁₆	
0396₁₆	タイマA0モードレジスタ(TA0MR)
0397₁₆	タイマA1モードレジスタ(TA1MR)
0398₁₆	タイマA2モードレジスタ(TA2MR)
0399₁₆	タイマA3モードレジスタ(TA3MR)
039A₁₆	タイマA4モードレジスタ(TA4MR)
039B₁₆	タイマB0モードレジスタ(TB0MR)
039C₁₆	タイマB1モードレジスタ(TB1MR)
039D₁₆	タイマB2モードレジスタ(TB2MR)
039E₁₆	
039F₁₆	
03A0₁₆	UART0送受信モードレジスタ(U0MR)
03A1₁₆	UART0転送速度レジスタ(U0BRG)
03A2₁₆	UART0送信バッファレジスタ(U0TB)
03A3₁₆	
03A4₁₆	UART0送受信制御レジスタ０(U0C0)
03A5₁₆	UART0送受信制御レジスタ１(U0C1)
03A6₁₆	UART0受信バッファレジスタ(U0RB)
03A7₁₆	
03A8₁₆	UART1送受信モードレジスタ(U1MR)
03A9₁₆	UART1転送速度レジスタ(U1BRG)
03AA₁₆	UART1送信バッファレジスタ(U1TB)
03AB₁₆	
03AC₁₆	UART1送受信制御レジスタ０(U1C0)
03AD₁₆	UART1送受信制御レジスタ１(U1C1)
03AE₁₆	UART1受信バッファレジスタ(U1RB)
03AF₁₆	
03B0₁₆	UART送受信制御レジスタ２(UCON)
03B1₁₆	
03B2₁₆	
03B3₁₆	
03B4₁₆	
03B5₁₆	
03B6₁₆	
03B7₁₆	
03B8₁₆	DMA0要因選択レジスタ(DM0SL)
03B9₁₆	
03BA₁₆	DMA1要因選択レジスタ(DM1SL)
03BB₁₆	
03BC₁₆	CRCデータレジスタ(CRCD)
03BD₁₆	
03BE₁₆	CRCインプットレジスタ(CRCIN)
03BF₁₆	

03C0₁₆	A-Dレジスタ0(AD0)
03C1₁₆	
03C2₁₆	A-Dレジスタ1(AD1)
03C3₁₆	
03C4₁₆	A-Dレジスタ2(AD2)
03C5₁₆	
03C6₁₆	A-Dレジスタ3(AD3)
03C7₁₆	
03C8₁₆	A-Dレジスタ4(AD4)
03C9₁₆	
03CA₁₆	A-Dレジスタ5(AD5)
03CB₁₆	
03CC₁₆	A-Dレジスタ6(AD6)
03CD₁₆	
03CE₁₆	A-Dレジスタ7(AD7)
03CF₁₆	
03D0₁₆	
03D1₁₆	
03D2₁₆	
03D3₁₆	
03D4₁₆	A-D制御レジスタ2(ADCON2)
03D5₁₆	
03D6₁₆	A-D制御レジスタ0(ADCON0)
03D7₁₆	A-D制御レジスタ1(ADCON1)
03D8₁₆	D-Aレジスタ0(DA0)
03D9₁₆	
03DA₁₆	D-Aレジスタ1(DA1)
03DB₁₆	
03DC₁₆	D-A制御レジスタ(DACON)
03DD₁₆	
03DE₁₆	
03DF₁₆	
03E0₁₆	ポートP0(P0)
03E1₁₆	ポートP1(P1)
03E2₁₆	ポートP0方向レジスタ(PD0)
03E3₁₆	ポートP1方向レジスタ(PD1)
03E4₁₆	ポートP2(P2)
03E5₁₆	ポートP3(P3)
03E6₁₆	ポートP2方向レジスタ(PD2)
03E7₁₆	ポートP3方向レジスタ(PD3)
03E8₁₆	ポートP4(P4)
03E9₁₆	ポートP5(P5)
03EA₁₆	ポートP4方向レジスタ(PD4)
03EB₁₆	ポートP5方向レジスタ(PD5)
03EC₁₆	ポートP6(P6)
03ED₁₆	ポートP7(P7)
03EE₁₆	ポートP6方向レジスタ(PD6)
03EF₁₆	ポートP7方向レジスタ(PD7)
03F0₁₆	ポートP8(P8)
03F1₁₆	ポートP9(P9)
03F2₁₆	ポートP8方向レジスタ(PD8)
03F3₁₆	ポートP9方向レジスタ(PD9)
03F4₁₆	ポートP10(P10)
03F5₁₆	
03F6₁₆	ポートP10方向レジスタ(PD10)
03F7₁₆	
03F8₁₆	
03F9₁₆	
03FA₁₆	
03FB₁₆	
03FC₁₆	プルアップ制御レジスタ０(PUR0)
03FD₁₆	プルアップ制御レジスタ１(PUR1)
03FE₁₆	プルアップ制御レジスタ２(PUR2)
03FF₁₆	

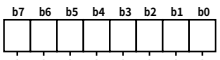
図 2.1.4　制御レジスタの配置 2

動作モードの決定

M16C/60 グループの動作モードは、プロセッサモードレジスタ 0（00004H 番地）のビット 0 とビット 1 で決定します。

図 2.1.5 に M16C/60 グループのプロセッサモードレジスタ 0 を示します。

プロセッサモードレジスタ 0 (注1)



シンボル　PM0　　アドレス　0004$_{16}$番地　　リセット時　00$_{16}$(注2)

ビットシンボル	ビット名	機　能	R	W
PM00	プロセッサモードビット	b1 b0 0 0 : シングルチップモード 0 1 : メモリ拡張モード 1 0 : 使用禁止 1 1 : マイクロプロセッサモード	○	○
PM01			○	○
PM02	R/Wモード選択ビット	0 : $\overline{RD}$,$\overline{BHE}$,$\overline{WR}$ 1 : $\overline{RD}$,$\overline{WRH}$,$\overline{WRL}$	○	○
PM03	ソフトウエアリセットビット	このビットに“1”を書き込むとマイクロコンピュータはリセットされる。 読み出し時の値は“0”。	○	○
PM04	マルチプレクスバス空間選択ビット	b5 b4 0 0 : マルチプレクスバスを使用しない 0 1 : CS2の空間に割り当てる 1 0 : CS1の空間に割り当てる 1 1 : 全空間に割り当てる(注4)	○	○
PM05			○	○
PM06	ポートP4$_0$～P4$_3$機能選択ビット(注3)	0 : アドレス出力 1 : ポート機能 (アドレスは出力されません)	○	○
PM07	BCLK出力禁止ビット	0 : 出力する 1 : 出力しない (端子はフローティングになります)	○	○

注1. このレジスタを書き替える場合、プロテクトレジスタ(000A$_{16}$番地)のビット1を“1”にしてください。
注2. CNVss端子にVccレベルを印加しているときは、リセット時03$_{16}$になります(PM00およびPM01が“1”になります)。
注3. マイクロプロセッサモード、メモリ拡張モード時有効。
注4. マイクロプロセッサモード時、全空間マルチプレクスバスは選択できません。メモリ拡張モード時、全空間マルチプレクスバスを選択した場合、アドレスバスはチップセレクトごとに256バイトの範囲です。

図 2.1.5　プロセッサモードレジスタ 0

2.1.3 固定ベクタ領域

M16C/60 グループの固定ベクタ領域は FFE00H 番地から FFFFFH 番地です。

固定ベクタ領域の FFE00H 番地から FFFDBH 番地はスペシャルページベクタテーブルです。ここにサブルーチンの先頭番地またはジャンプ先を格納すると、サブルーチンコール命令やジャンプ命令を2バイトで実行でき、プログラムステップ数の節減に役立ちます。

固定ベクタ領域の FFFDCH 番地から FFFFFH 番地は、リセットおよびNMIなどの固定割り込みベクタテーブルです。ここに割り込みルーチンの先頭アドレスを格納します。なお、タイマ割り込みなどの割り込みベクタテーブルは、内部レジスタ(INTB)により任意の番地に設定することができます。詳しくは第4章　割り込みの節をご参照ください。

固定ベクタ領域のメモリ配置図

スペシャルページベクタテーブルと固定ベクタ領域のメモリ配置を図 2.1.6 に示します。

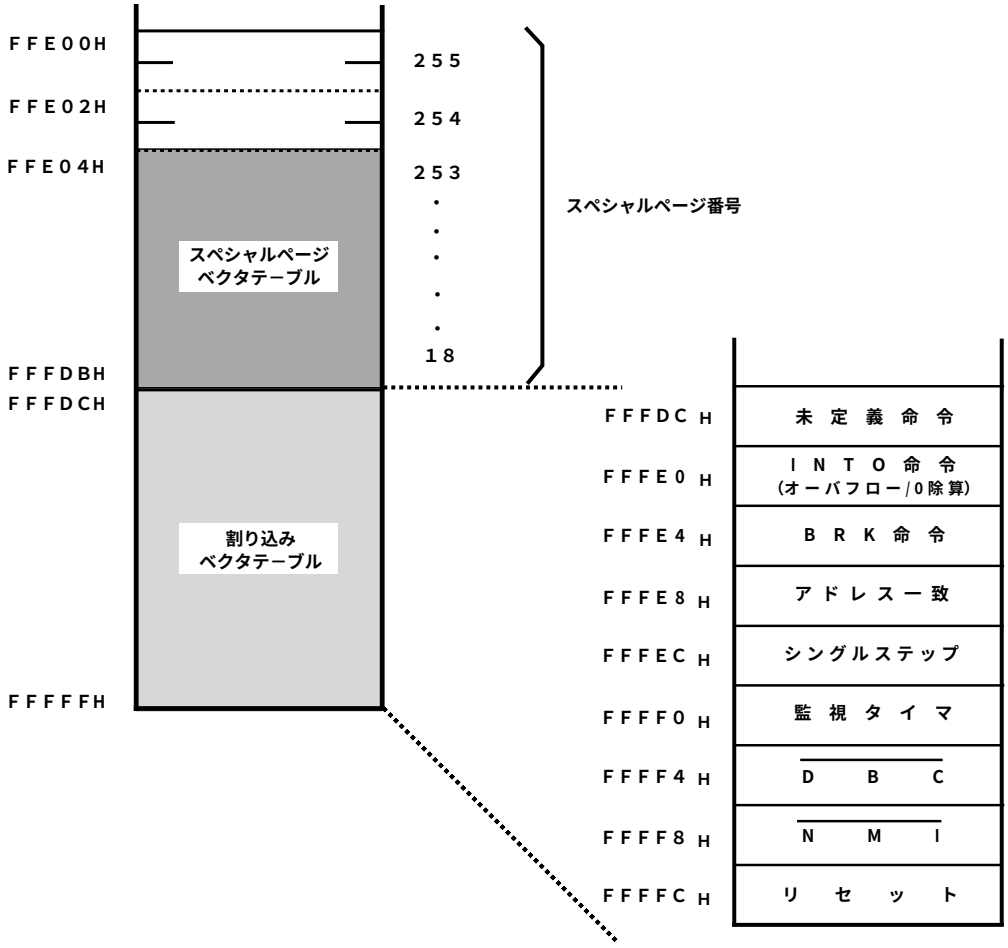


図 2.1.6　固定ベクタ領域のメモリ配置

2.2 レジスタセット

M16C/60 シリーズ CPU コアの汎用レジスタと専用レジスタについて説明します。

レジスタ構成

M16C/60 シリーズ CPU コアのレジスタ構成を図 2.2.1 に示します。レジスタ R0、R1、R2、R3、A0、A1、FB の 7 本は 2 セット用意されています。各レジスタの機能を以下に示します。

汎用レジスタ

(1)データレジスタ(R0、R1、R2、R3)

データレジスタ(R0、R1、R2、R3)は 16 ビットで構成されており、主に転送や算術、論理演算に使用します。

R0 と R1 は、上位(R0H、R1H)と下位(R0L、R1L)を別々に 8 ビットのデータレジスタとして使用することもできます。また、一部の命令では R2 と R0、R3 と R1 を組み合わせて 32 ビットのデータレジスタ(R2R0、R3R1)としても使用できます。

(2)アドレスレジスタ(A0、A1)

アドレスレジスタ(A0、A1)は16ビットで構成されており、データレジスタと同等の機能を持ちます。また、アドレスレジスタ間接アドレッシングおよび、アドレスレジスタ相対アドレッシングに使用します。

一部の命令では A1 と A0 を組み合わせて 32 ビットのアドレスレジスタ(A1A0)としても使用できます。

(3)フレームベースレジスタ(FB)

フレームベースレジスタ(FB)は 16 ビットで構成されており、FB 相対アドレッシングに使用します。

(4)スタティックベースレジスタ(SB)

スタティックベースレジスタ(SB)は16ビットで構成されており、SB相対アドレッシングに使用します。

専用レジスタ

(5)プログラムカウンタ(PC)

プログラムカウンタ(PC)は 20 ビットで構成されており、実行する命令の番地を示します。

(6)割り込みテーブルレジスタ(INTB)

割り込みテーブルレジスタ(INTB)は20ビットで構成されており、割り込みベクタテーブルの先頭番地を示します。

(7)スタックポインタ(USP、ISP)

スタックポインタは、ユーザスタックポインタ(USP)と割り込みスタックポインタ(ISP)の2種類があり、共に 16 ビットで構成されています。

使用するスタックポインタ(USP、ISP)はスタックポインタ指定フラグ(U フラグ)によって切り換えられます。U フラグは、フラグレジスタ(FLG)のビット 7 です。

(8)フラグレジスタ(FLG)

フラグレジスタ(FLG)は 11 ビットで構成されており、1 ビット単位でフラグとして使用します。

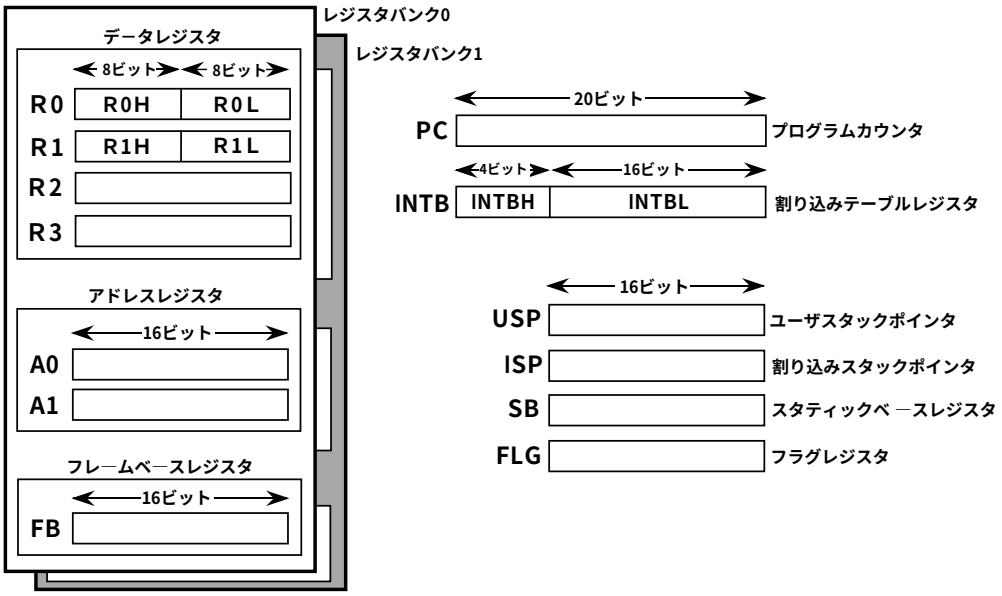


図 2.2.1　レジスタ構成

フラグレジスタ(FLG)

フラグレジスタ(FLG)のビット構成を図 2.2.2 に示します。各フラグの機能を以下に示します。

●ビット 0:キャリーフラグ(C フラグ)
算術、論理演算で発生したキャリー、ボロー、シフトアウトしたビットなどを保持します。

●ビット 1:デバッグフラグ(D フラグ)
シングルステップ割り込みを許可するフラグです。
このフラグが“ 1 ”のとき、命令実行後シングルステップ割り込みが発生します。割り込みを受け付けるとこのフラグは“ 0 ”になります。

●ビット 2:ゼロフラグ(Z フラグ)
演算結果が 0 のとき“ 1 ”になり、それ以外のとき“ 0 ”になります。

●ビット 3:サインフラグ(S フラグ)
演算結果が負のとき“ 1 ”になり、正のとき“ 0 ”になります。

●ビット 4:レジスタバンク指定フラグ(B フラグ)
レジスタバンクの選択を行います。このフラグが“ 0 ”のときレジスタバンク 0 が指定され、“ 1 ”のときレジスタバンク 1 が指定されます。

●ビット 5:オーバフローフラグ(O フラグ)
演算結果がオーバフローしたとき“ 1 ”になります。

●ビット 6:割り込み許可フラグ(I フラグ)
マスカブル割り込みを許可するフラグです。
このフラグが“ 1 ”のとき割り込みは許可され、“ 0 ”のとき割り込みは禁止されます。割り込みを受け付けるとこのフラグは“ 0 ”になります。

●ビット 7:スタックポインタ指定フラグ(U フラグ)
このフラグが“ 1 ”のときユーザスタックポインタ(USP)が指定され、“ 0 ”のとき割り込みスタックポインタ（ISP）が指定されます。
ハードウェア割り込みを受け付けたとき、またはソフトウェア割り込み番号 0 ～ 31 の INT 命令を実行したとき、このフラグは“ 0 ”になります。

●ビット 8 ～ビット 11:予約領域

●ビット 12 ～ビット 14:プロセッサ割り込み優先レベル(IPL)
プロセッサ割り込み優先レベル(IPL)は 3 ビットで構成され、レベル 0 からレベル 7 までの 8 段階の IPL を指定します。
要求があった割り込みの優先レベルが、IPL より大きい場合、その割り込みは許可されます。

●ビット 15:予約領域

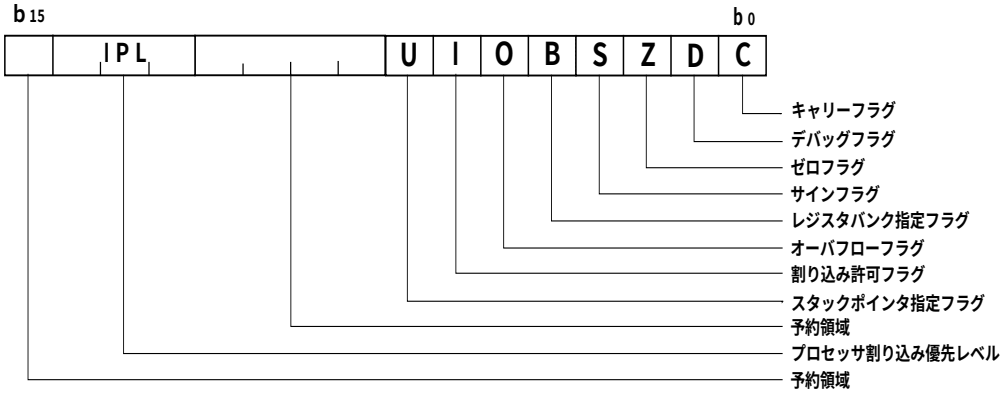


図 2.2.2　フラグレジスタ(FLG)のビット構成

リセット解除後のレジスタの状態

リセット解除後の各レジスタの表 2.2.1 に示します。（注）

表 2.2.1　リセット解除後のレジスタの状態

レジスタ名	リセット解除後の状態
データレジスタ（R0/R1/R2/R3）	0000H
アドレスレジスタ（A0/A1）	0000H
フレームベースレジスタ（FB）	0000H
割り込みテーブルレジスタ（INTB）	00000H
ユーザスタックポインタ（USP）	0000H
割り込みスタックポインタ（ISP）	0000H
スタティックベースレジスタ（SB）	0000H
フラグレジスタ（FLG）	0000H

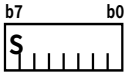
2.3 データタイプ

M16C/60 シリーズ、M16C/20 シリーズで取り扱うデータタイプは整数、10 進(BCD)、ストリング、ビットの 4 種類です。ここではデータタイプについて説明します。

整数

整数は符号付きと符号なしがあります。符号付き整数の負の値は 2 の補数で表現します。

符号付きバイト(8ビット)整数



符号なしバイト(8ビット)整数

符号付きワード(16ビット)整数

符号なしワード(16ビット)整数

符号付きロングワード(32ビット)整数

符号なしロングワード(32ビット)整数

図 2.3.1 整数データ

S : 符号ビット

10 進(BCD)

BCD コードはパック形式で取り扱います。
10 進演算命令 DADC、DADD、DSBB、DSUB の 4 種類の命令で使用できます。

1バイトパック形式
(2桁)

2バイトパック形式
(4桁)

図 2.3.2 10 進データ

ストリング

1バイトまたは1ワード(16ビット)のデータを任意の数だけ連続して並べたデータのブロックをストリングといいます。

ストリング命令 SMOVB、SMOVF、SSTR の 3 種類の命令で使用できます。

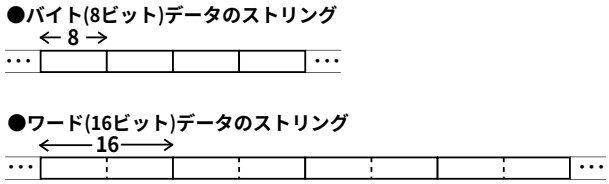


図 2.3.3　ストリングデータ

ビット

ビットはビット命令 BCLR、BSET、BTST、BNTST などの 14 種類の命令で使用できます。各レジスタのビットは、レジスタ名と 0～15 のビット番号で指定します。メモリのビットはアドレッシングモードにより、指定方法、指定範囲が異なります。　使い方については、「2.5.4　ビット命令アドレッシング」をご参照ください。

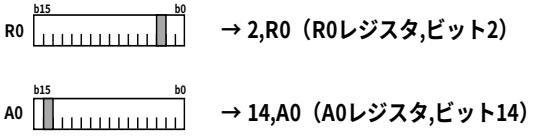


図 2.3.4　レジスタのビット指定

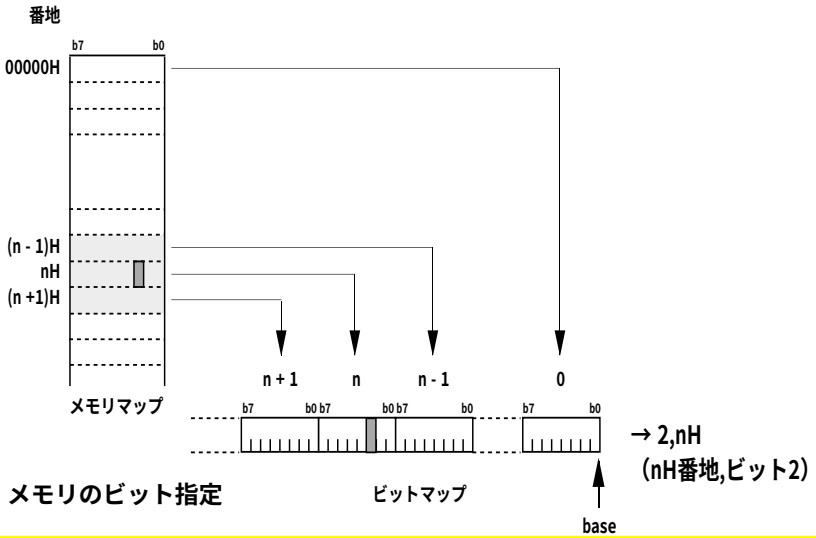


図 2.3.5　メモリのビット指定

2.4 データ配置

M16C/60シリーズ、M16C/20シリーズでは、ニブル(4ビット)やバイト(8ビット)のデータも効率良く取り扱うことができます。ここでは取り扱うことのできるデータ配置について説明します。

レジスタのデータ配置

レジスタのデータサイズとビット番号の関係を図 2.4.1 に示します。
図のように、最下位ビット(LSB)のビット番号は 0 になります。最上位ビット(MSB)のビット番号は取り扱うデータサイズによって異なります。

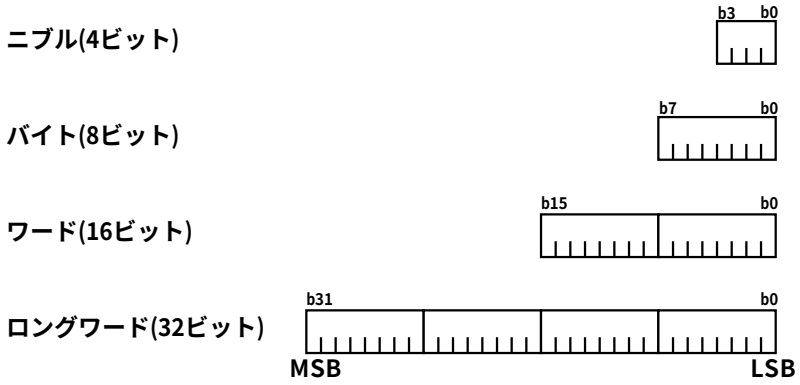


図 2.4.1 レジスタのデータ配置

メモリ上のデータ配置

M16C/60 シリーズ、M16C/20 シリーズメモリ上のデータ配置を図 2.4.2 に示します。
メモリでは図のように 8 ビット単位でデータを配置します。ワード(16 ビット)は下位バイト、上位バイトに分けて、下位バイト:DATA(L)を小さい番地に配置します。アドレス(20 ビット)、ロングワード(32 ビット)も同様に下位バイト:DATA(L),DATA(LL)からメモリ上に配置します。

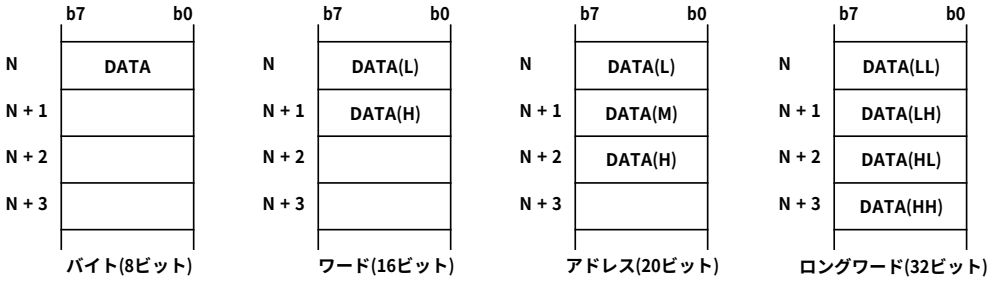


図 2.4.2 メモリ上のデータ配置

2.5　アドレッシングモード

M16C/60 シリーズ、M16C/20 シリーズのアドレッシングについて説明します。

2.5.1　アドレッシングモードの種類

アドレッシングモードは以下に示す 3 つのタイプに分けられます。

(1)一般命令アドレッシング

00000H 番地から 0FFFFH 番地までの領域をアクセスします。

(2)特定命令アドレッシング

00000H 番地から FFFFFH 番地までのアドレス空間全体をアクセスします。

(3)ビット命令アドレッシング

00000H 番地から 0FFFFH 番地までの領域をビット単位でアクセスします。

このアドレッシングモードはビット命令で使用します。

アドレッシングモード一覧

全アドレッシングを表 2.5.1 にまとめます。

表 2.5.1　M16C/60 シリーズ、M16C/20 シリーズのアドレッシングモード

項目	内　容		
アドレッシングモード	一般命令	特定命令	ビット命令
即値	○　imm : 8/16ビット	×	×
レジスタ直接	○ データレジスタ, アドレスレジスタのみ	○　R2R0 or R3R1 or A1A0 ※SHL, SHA, JMPI, JSRI命令のみ	○　R0 , R1 , R2 , R3 , A0 , A1のみ
絶対	○　abs : 16ビット(0〜FFFFH)	○　abs : 20ビット(0〜FFFFFH) ※LDE, STE, JMP, JSR命令のみ	○　bit,base : 16ビット(0〜1FFFH)
アドレスレジスタ間接	○　［A0］or［A1］dspなし	○　［A1A0］dspなし ※LDE, STE命令のみ	○　［A0］or［A1］dspなし (0〜1FFFH)
アドレスレジスタ相対	○　［A0］or［A1］dsp : 8/16ビット	○　［A0］dsp : 20ビットのみ ※LDE, STE, JMPI, JSRI 命令のみ	○　［A0］or［A1］ dsp : 8/16ビット
SB相対とFB相対	○　［SB］dsp : 8/16bit (0〜255 / 0〜65534) ○　［FB］dsp: 8bit(-128〜+127)	×	○　［SB］dsp : 8/11/16bit (0〜31 / 0〜255 / 0〜8191) ○　［FB］dsp: 8bit(-16〜+15)
スタックポインタ相対	○［SP］dsp : 8ビット(-128〜+127) ※MOV命令のみ	×	×
プログラムカウンタ相対	×	○　label　.S : +2 〜 +9 :　.B : -128 〜 +127 .W : -32768 〜 +32767 ※JMP, JSR命令のみ	×
専用レジスタ直接	×	○ INTBL, INTBH, ISP, USP, SB, FB, FLG ※LDC, STC, PUSHC, POPC命令のみ	×
ＦＬＧ直接	×	×	○　U, I, O, B, S, Z, D, Cﾌﾗｸﾞ ※FCLR, FSET命令のみ

2.5.2　一般命令アドレッシング

この項では一般命令アドレッシングの各アドレッシングについて説明します。

即値

#IMM で示した即値が演算対象になります。即値の前に # を付けてください
(記号)#IMM、#IMM8、#IMM16、#IMM20
(例)#123(10 進数)
#7DH(16 進数)
#01111011B(2 進数)

絶対

abs16で指定した値が演算対象の実効アドレスになります。実効アドレスの範囲は00000H～0FFFFH番地です。
(記号)abs16
(例)8000H
DATA(ラベル)

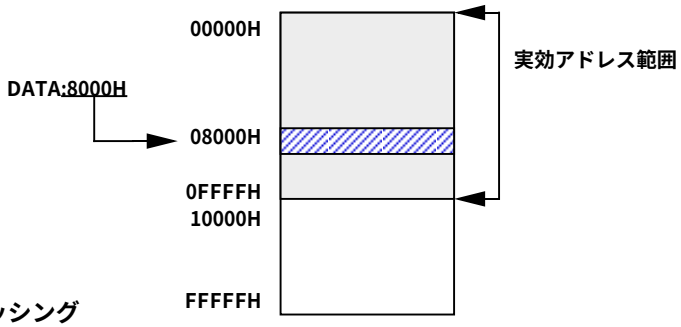


図 2.5.1　絶対アドレッシング

レジスタ直接

指定したレジスタが演算対象になります。
ただし、データレジスタ、アドレスレジスタのみ使用できます。
(記号) 8 ビット　R0L、R0H、R1L、R1H
16 ビット　R0、R1、R2、R3、A0、A1

アドレスレジスタ間接

アドレスレジスタの値が実効アドレスとなります。実効アドレスの範囲は 00000H ～ 0FFFFH 番地です
(記号)[A0]、[A1]
(例)MOV.B　#12H,[A0]

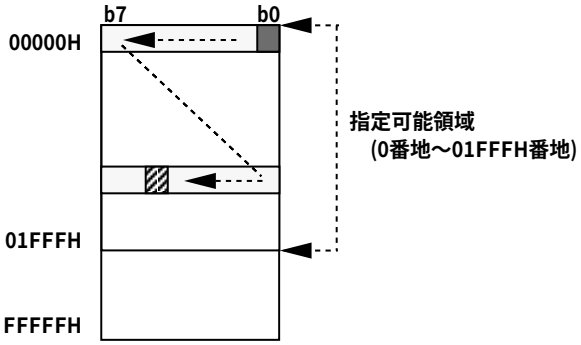


図 2.5.2　アドレスレジスタ間接アドレッシング

アドレスレジスタ相対

アドレスレジスタとディスプレースメント(dsp)(注)を加算した値が実効アドレスとなります。実効アドレスの範囲は 00000H ～ 0FFFFH 番地です。加算結果が 0FFFFH を越えた場合、17 ビット以上は無視されます。

(記号)dsp:8[A0]、dsp:16[A0]、dsp:8[A1]、dsp:16[A1]

(1)dsp を変位として扱った場合

(例)MOV.B　#34H,5[A0]

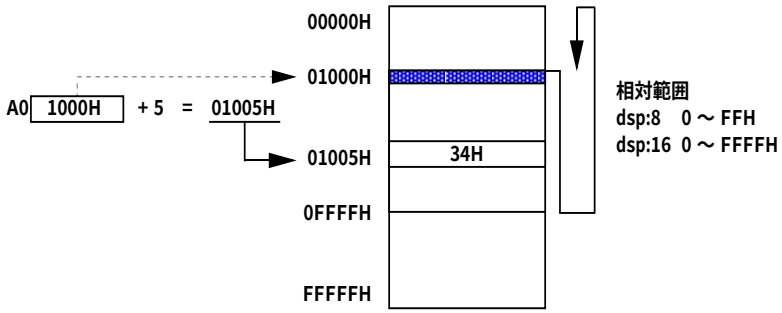


図 2.5.3　アドレスレジスタ相対アドレッシング 1

(2)アドレスレジスタ(A0)を変位として扱った場合

(例)MOV.B　#56H,1234H[A0]

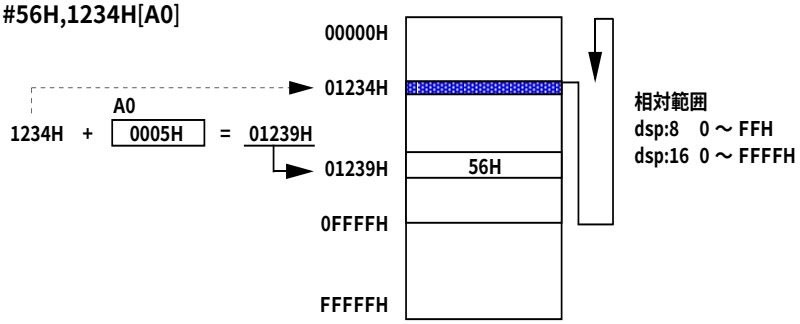


図 2.5.4　アドレスレジスタ相対アドレッシング 2

(3)加算結果が 0FFFFH を越えた場合

（例）MOV. B　#56H,1234H[A0]

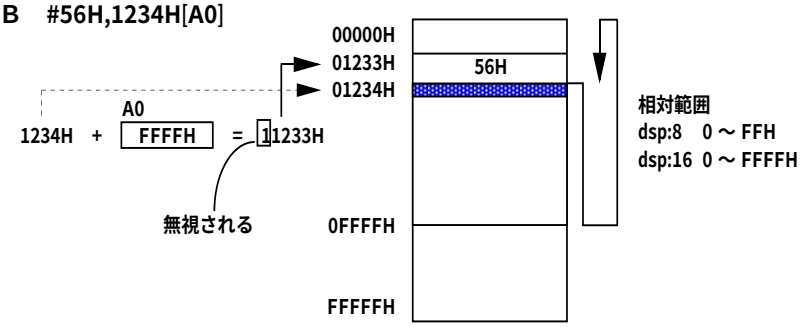


図 2.5.5　アドレスレジスタ相対アドレッシング 3

（注）ディスプレースメント(dsp)とは基準アドレスからの変位です。本書では 8 ビットの dsp を dsp:8、16 ビットの dsp を dsp:16 と表現します。

SB 相対

SB レジスタと dsp を加算した値が実効アドレスとなります。実効アドレスの範囲は 00000H ～ 0FFFFH 番地です。加算結果が 0FFFFH を越えた場合、17 ビット以上は無視されます。

(記号)dsp:8[SB]、dsp:16[SB]

(例)MOV.B　#12H,5[SB]

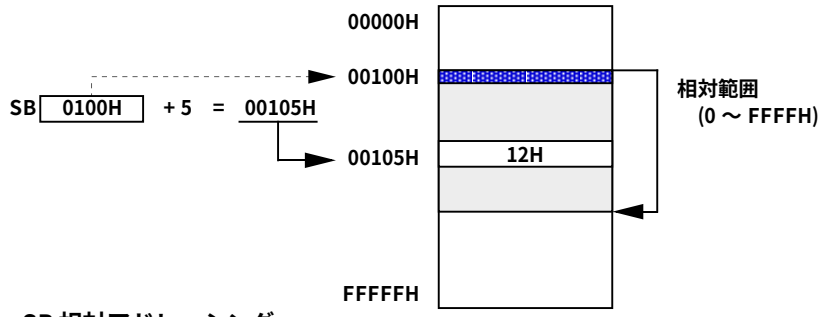


図 2.5.6　SB 相対アドレッシング

FB 相対

FB レジスタと dsp を加算した値が実効アドレスとなります。実効アドレスの範囲は 00000H ～ 0FFFFH 番地です。加算結果が 0FFFFH を越えた場合、17 ビット以上は無視されます。

(記号)dsp:8[FB]

(1)dsp の値が正のとき

(例)MOV.B　#12H,5[FB]

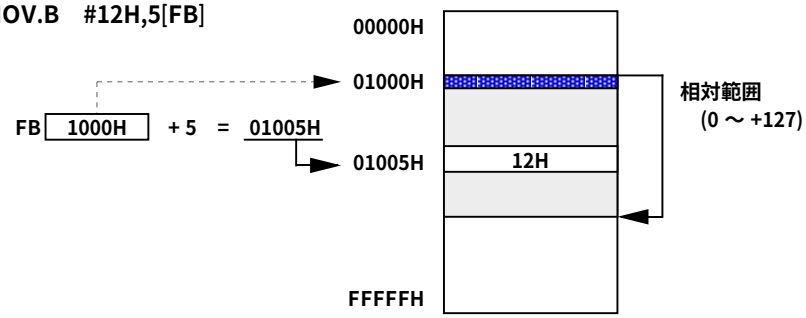


図 2.5.7　FB 相対アドレッシング 1

(2)dsp の値が負のとき

(例)MOV.B　#12H,-5[FB]

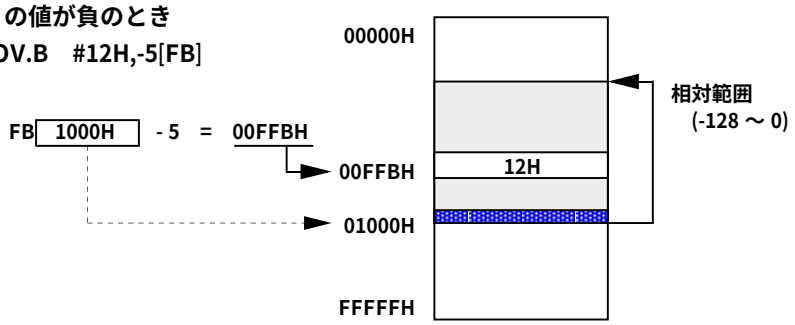


図 2.5.8　FB 相対アドレッシング 2

コラム　SB 相対と FB 相対の違い

SB 相対アドレッシングは SB レジスタと dsp を加算した値を実効アドレスとするアドレッシングです。相対範囲は dsp:8[SB]の場合は 0 ～ +255(FFH)で、dsp:16[SB]の場合は 0 ～ +65535(FFFFH)です。

FB 相対アドレッシングは FB レジスタと dsp を加算した値、または FB レジスタから dsp を減算した値を実効アドレスとするアドレッシングです。相対範囲は -128 ～ +127(80H ～ 7FH)です。FB 相対は負方向にアクセスすることができます。dsp は 8 ビットのみで、16 ビットは使えません。

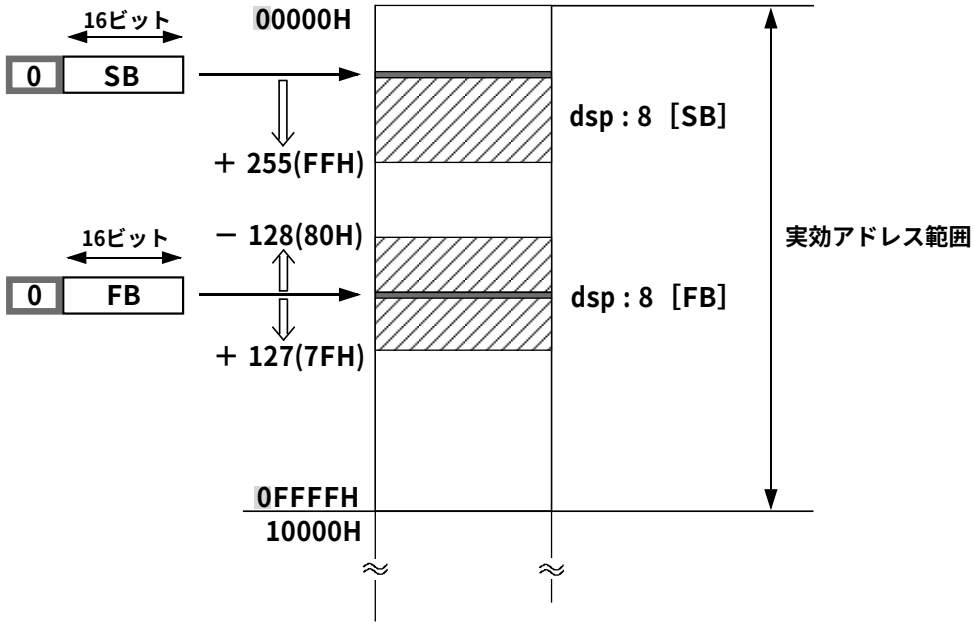


図 2.5.9　SB 相対アドレッシングと FB 相対アドレッシング

コラム　SB 相対の応用例

SB相対アドレッシングは図2.5.10のようにタスクの固有データテーブルへ応用できます。各タスクを動作させるために必要なデータはタスクのスイッチング時に切り替えますが、SB相対アドレッシングを使うとSBレジスタを書き替えるだけで簡単に切り替えることができます。

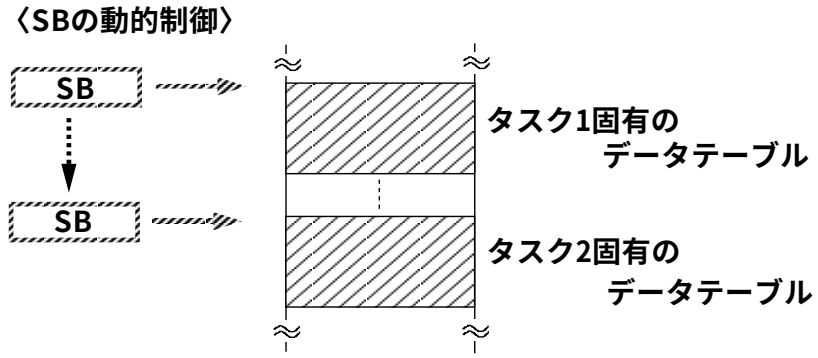


図 2.5.10　SB 相対の応用例

コラム　FB 相対の応用例

FB相対アドレッシングは図2.5.11のように関数を呼び出したときのスタックフレームへ応用できます。スタックフレームでは、マイナス方向のアドレスにローカル変数領域が配置されるので、ベースからプラス方向もマイナス方向もアクセスできるFB相対アドレッシングが必要になります。

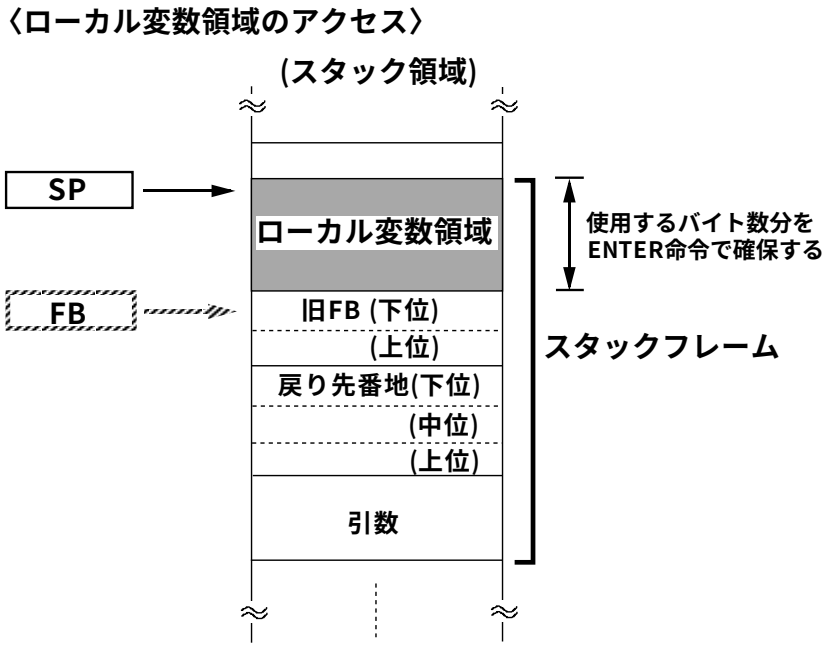


図 2.5.11　FB 相対の応用例

スタックポインタ相対(SP 相対)

SP 相対アドレッシングは、SP と dsp を加算した値、または SP レジスタから dsp を減算した値が実効アドレスとなります。SP 相対アドレッシングは MOV 命令でのみ使用できます。なお、即値の転送はできません。実効アドレスの範囲は 00000H ～ 0FFFFH 番地です。加算結果が 0FFFFH を越えた場合、17 ビット以上は無視されます。

(記号)dsp:8[SP]

(1)dsp の値が正のとき

(例)MOV.B　R0L,5[SP]

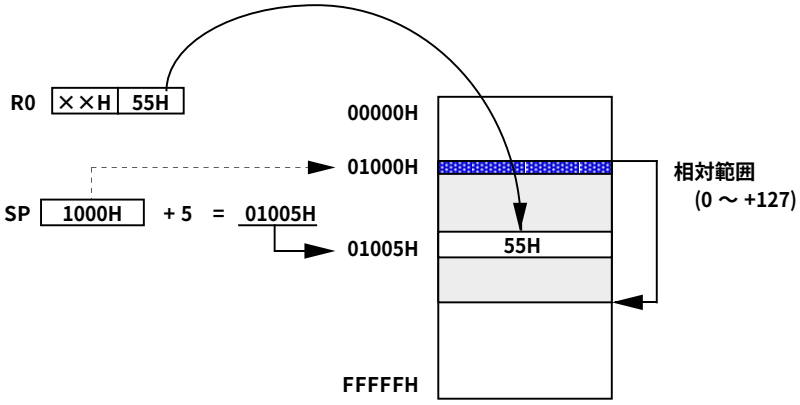


図 2.5.12　SP 相対アドレッシング 1

(2)dsp の値が負のとき

(例)MOV.B　R0L,-5[SP]

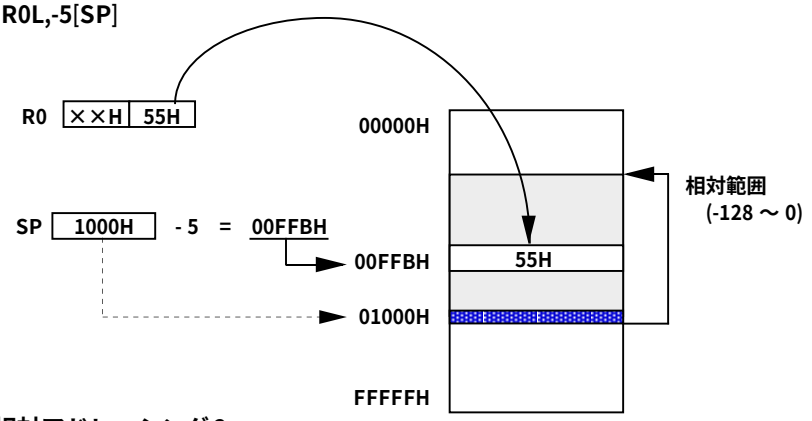


図 2.5.13　SP 相対アドレッシング 2

コラム　相対アドレッシングの相対範囲

相対アドレッシングの相対範囲を表 2.5.2 にまとめます。

表 2.5.2　相対アドレッシングの相対範囲

アドレッシングモード	記述形式	相対範囲
アドレスレジスタ相対	dsp:8[An] dsp:16[An] dsp:20[An](注)	0～255(0FFH) 0～65535(0FFFFH) 0～1048575(0FFFFFH)
SB相対とFB相対	dsp:8[SB] dsp:16[SB] dsp:8[FB]	0～255(0FFH) 0～65535(0FFFFH) -128(80H)～+127(7FH)
スタックポインタ相対	dsp:8[SP]	-128(80H)～+127(7FH)

（注）dsp:20[An]は、LDE、STE、JMPI、JSRI 命令で使用できます。

2.5.3 特定命令アドレッシング

特定命令アドレッシングは 00000H ～ FFFFFH までのアドレス空間をアクセスできます。この項では、特定命令アドレッシングの各アドレッシングについて説明します。

20 ビット絶対

指定する20ビットの値が実効アドレスとなります。実効アドレスの範囲は00000H～FFFFFH番地です。20ビット絶対アドレッシングは LDE、STE、JMP、JSR 命令で使用できます。

(記号)abs20
(例)LDE.B　DATA,R0L

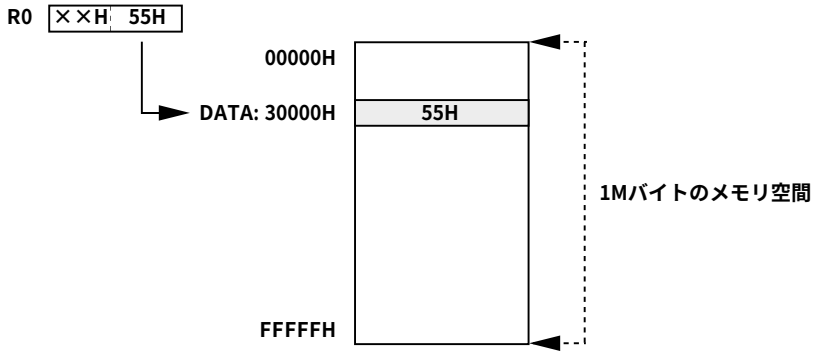


図 2.5.14　20 ビット絶対アドレッシング

32 ビットレジスタ直接

16 ビットレジスタを 2 個連結した、32 ビットのレジスタが演算の対象となります。SHL(論理シフト)、SHA(算術シフト)命令では R2R0、R3R1 が使用できます。また、JMPI(間接ジャンプ)、JSRI(間接サブルーチンコール)命令では R2R0、R3R1、A1A0 が使用できます。

(記号) R2R0、R3R1、A1A0

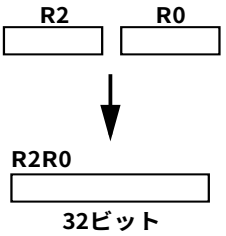


図 2.5.15　32 ビットレジスタ

(例) SHL.L　#4,R2R0 ------ R2R0の32ビット長の値を4ビット分左にシフトする。

シフト回数

(例) JMPI.A　R2R0 ------ R2R0の値が示す実効アドレス(20000H番地)に分岐する。

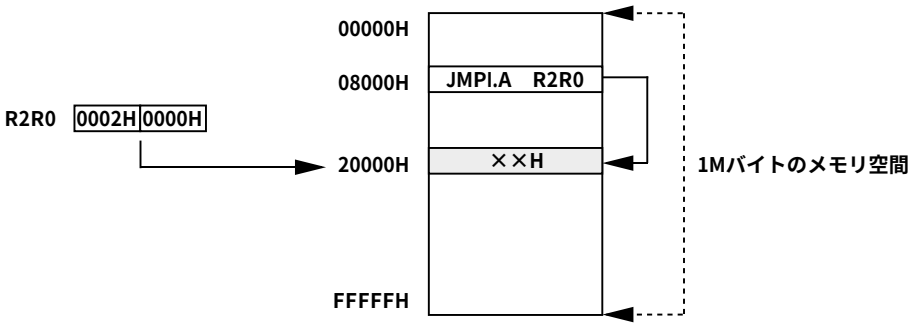


図 2.5.16　32 ビットレジスタ直接アドレッシング

専用レジスタ直接

専用レジスタをアクセスするアドレッシングです。このアドレッシングは LDC、STC、PUSHC、POPC 命令で使用できます。

(記号)INTBL、INTBH、ISP、SP(注)、SB、FB、FLG

(注)SP を指定した場合 U フラグで示すスタックポインタが対象となります。

32 ビットアドレスレジスタ間接

アドレスレジスタを 2 個連結した 32 ビットの値が実効アドレスとなります。実効アドレスの範囲は 00000H～FFFFFH 番地です。連結したレジスタの値が FFFFFH を越えた場合、21 ビット以上は無視されます。このアドレッシングは LDE、STE 命令で使用できます。

(記号)[A1A0]

(例)LDE.B　[A1A0],R0L

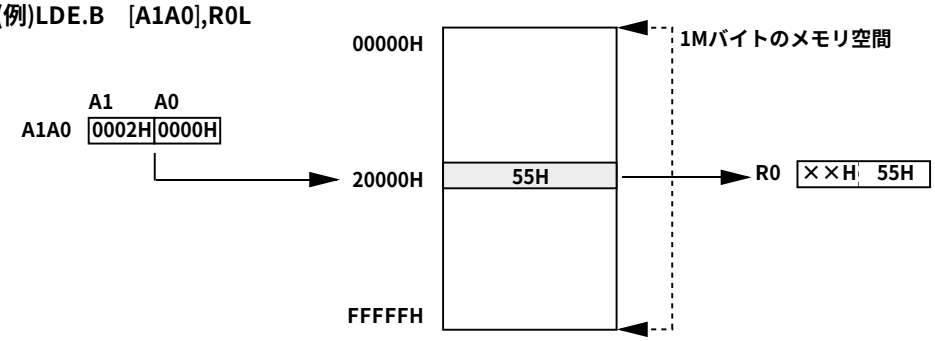


図 2.5.17　32 ビットアドレスレジスタ間接アドレッシング

20 ビットディスプレースメント付きアドレスレジスタ相対

アドレスレジスタと dsp を加算した値が実効アドレスとなります。実効アドレスの範囲は 00000H～FFFFFH 番地です。加算結果が FFFFFH を越えた場合、21 ビット以上は無視されます。このアドレッシングは LDE、STE、JMPI、JSRI 命令で使用できます。

(記号)dsp:20[A0]、dsp:20[A1]

(1)LDE/STE 命令の場合

(例)LDE.B　40000H[A0]，R0L

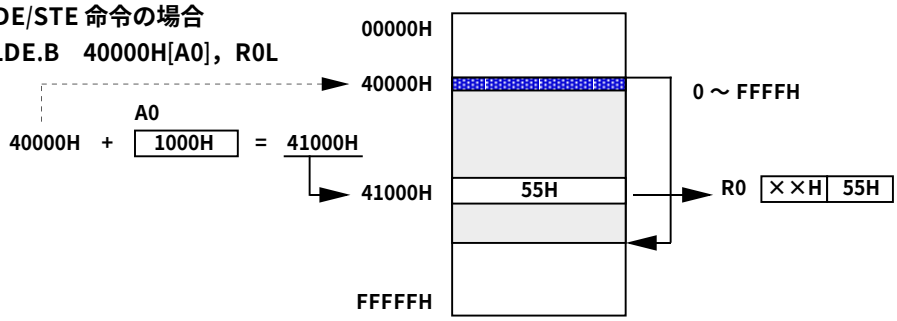


図 2.5.18　20 ビット dsp 付きアドレスレジスタ相対アドレッシング 1

(2)JMPI/JSRI 命令の場合

(例)JMPI.A　40000H[A0]

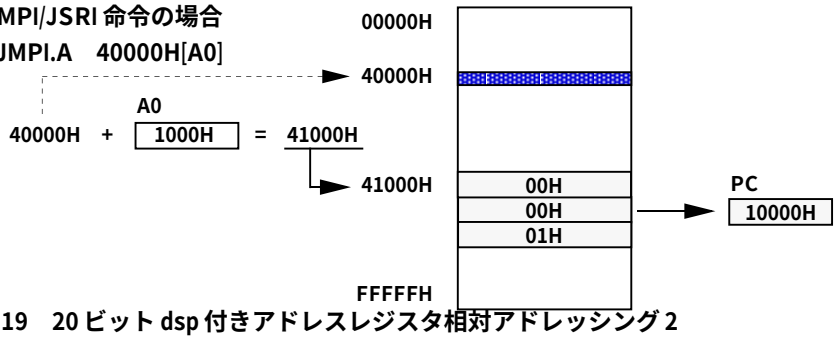


図 2.5.19　20 ビット dsp 付きアドレスレジスタ相対アドレッシング 2

PC 相対

プログラムカウンタ(PC)にdspを加算した値が実効アドレスとなります。ここでいうPCの値はアドレッシングを使用する命令の先頭番地です。PC 相対アドレッシングは JMP、JSR 命令で使用できます。

(1)分岐距離指定子(.length)が .S の場合
(記号)label　(PC+2 ≦ label ≦ PC+9)

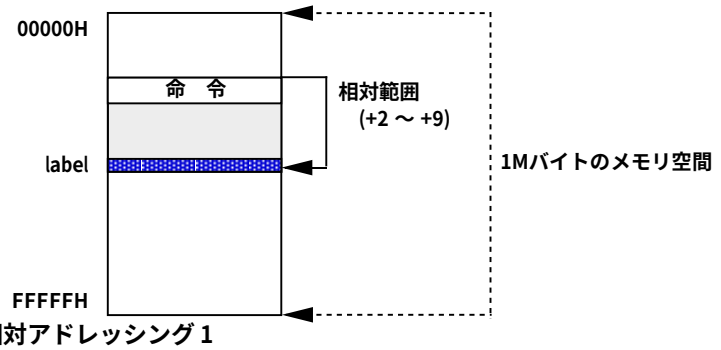


図 2.5.20　PC 相対アドレッシング 1

(2)分岐距離指定子(.length)が .B の場合
(記号)label　(PC-128 ≦ label ≦ PC+127)

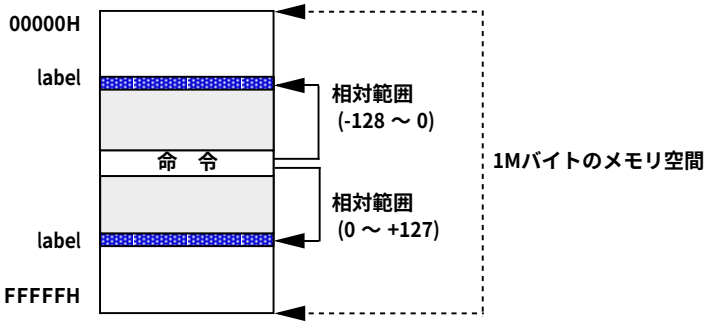


図 2.5.21　PC 相対アドレッシング 2

(3)分岐距離指定子(.length)が .W の場合
(記号)label　(PC-32768 ≦ label ≦ PC+32767)

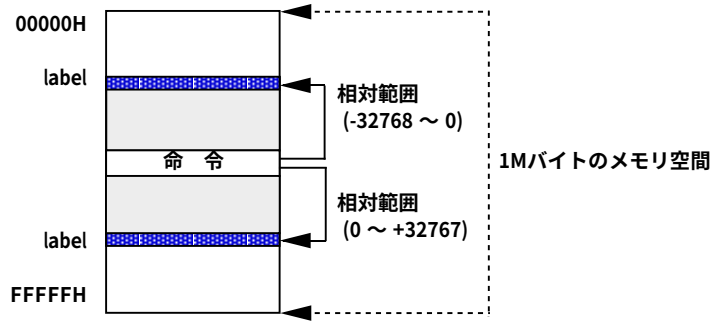


図 2.5.22　PC 相対アドレッシング 3

2.5.4　ビット命令アドレッシング

00000H ～ 0FFFFH までのアドレス空間をビット単位でアクセスします。
このアドレッシングはビット命令に使用します。この項ではビット命令アドレッシングの各アドレッシングについて説明します。

絶対

base で示したアドレスのビット 0 から bit で示したビット数分だけ離れたビットが演算対象となります。
指定可能領域は 00000H ～ 01FFFH 番地です。

(記号)bit,base16

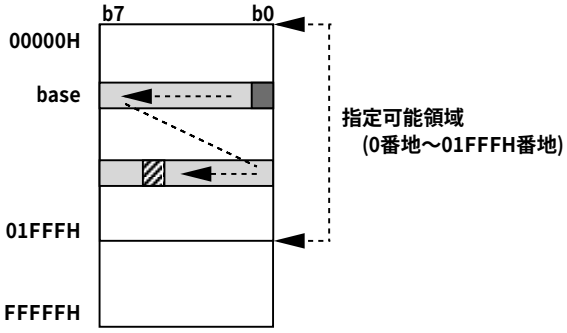


図 2.5.23　ビット命令絶対アドレッシング 1

(例 1)BCLR　18,base_addr
(例 2)BCLR　4,base_addr2
(例 3)BCLR　10,base_addr2　→例 3 は指定できません。

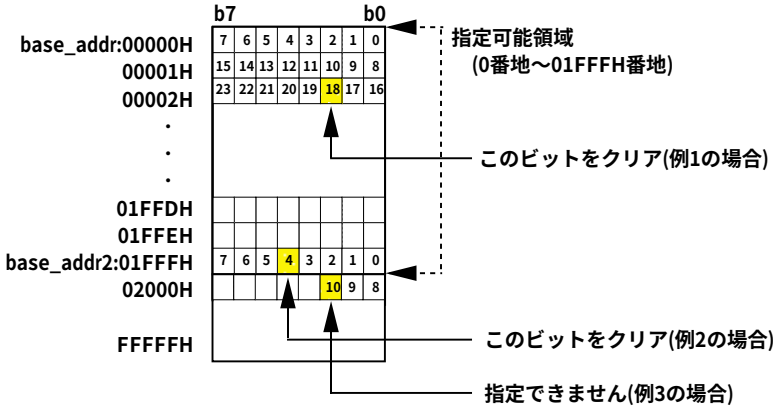


図 2.5.24　ビット命令絶対アドレッシング 2

レジスタ直接

16 ビットのレジスタ(R0、R1、R2、R3、A0、A1)のビットを指定します。
ビット位置は 0 ～ 15 で指定します。
(記号)bit,R0、bit,R1、bit,R2、bit,R3、bit,A0、bit,A1
(例)BCLR　6,R0

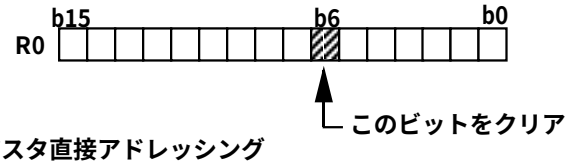


図 2.5.25　ビット命令レジスタ直接アドレッシング

FLG 直接

FCLR、FSET 命令で使用できます。ビット位置は FLG レジスタの下位 8 ビットのみ指定できます。

(記号)U、I、O、B、S、Z、D、C
(例)FSET　U

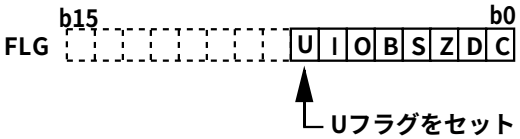


図 2.5.26　ビット命令 FLG 直接アドレッシング

アドレスレジスタ間接

00000H 番地のビット 0 から、アドレスレジスタ(A0、A1)で示したビット数だけ離れたビットが演算対象となります。

指定可能領域は 00000H ～ 01FFFH 番地です。

(記号)[A0]、[A1]

(例)BCLR　[A0]

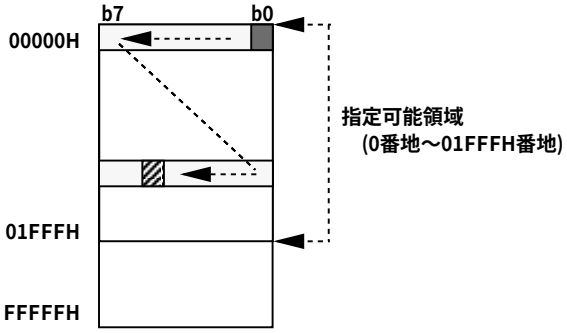


図 2.5.27　ビット命令アドレスレジスタ間接アドレッシング

アドレスレジスタ相対

base で示したアドレスのビット 0 から、アドレスレジスタ(A0、A1)で示したビット数だけ離れたビットが演算対象となります。

指定可能領域は base で示されるアドレスから 8K バイト(1FFFH)の領域です。ただし、実効アドレスの範囲は 00000H ～ 0FFFFH 番地です。対象となるビットのアドレスが 0FFFFH を越えた場合、17 ビット以上は無視されます。

(記号)base:8[A0]、base:16[A0]、base:8[A1]、base:16[A1]

(例)BCLR　5[A0]

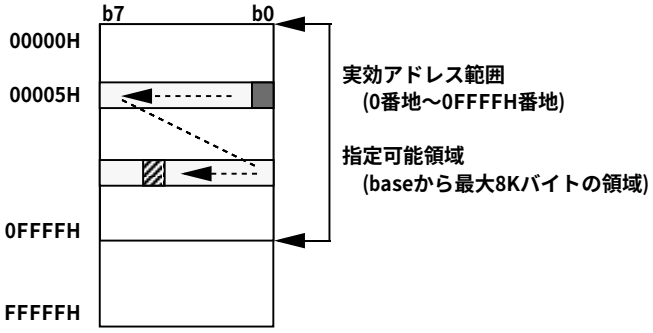


図 2.5.28　ビット命令アドレスレジスタ相対アドレッシング

SB 相対

基準はSBレジスタの値が示すアドレスです。SBレジスタの値にbaseを符号なしで加算します。その値が示すアドレスのビット0から、bitで示したビット数だけ離れたビットが演算対象となります。

指定可能領域はSBレジスタが示すアドレスから8Kバイトの領域です。ただし、実効アドレスの範囲は00000H～0FFFFH番地です。対象となるビットのアドレスが0FFFFHを越えた場合、17ビット以上は無視されます。

(記号)bit,base:8[SB]、bit,base:11[SB]、bit,base:16[SB]

注)bit,base :8[SB] : 最大 32 バイトの領域の 1 ビットを指定可能です。

bit,base:11[SB]: 最大 256 バイトの領域の 1 ビットを指定可能です。

bit,base:16[SB : 最大 8K バイトの領域の 1 ビットを指定可能です。

（例）BCLR 13,8[SB]

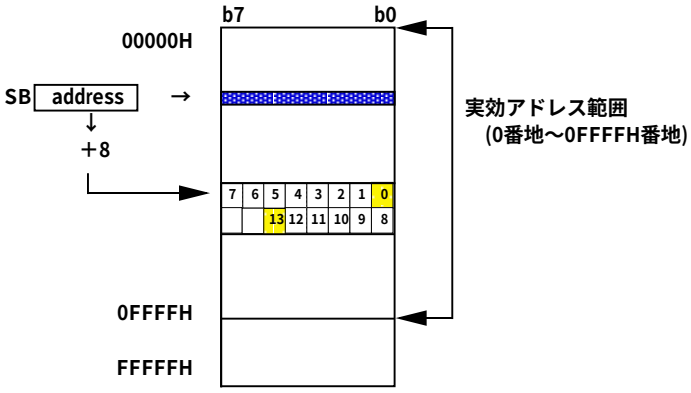


図 2.5.29　ビット命令 SB 相対アドレッシング

FB 相対

基準はFBレジスタの値が示すアドレスです。FBレジスタの値にbaseを符号付きで加算します。その値が示すアドレスのビット0から、bitで示したビット数だけ離れたビットが演算対象となります。

指定可能領域はFBレジスタが示すアドレスを中心にアドレスの小さい方に16バイト、大きい方に15バイトの領域です。ただし、実効アドレスの範囲は00000H～0FFFFH番地です。対象となるビットのアドレスが0FFFFHHを越えた場合、17ビット以上は無視されます。

(記号)bit,base:8[FB]
(例)BCLR　5,-8[FB]

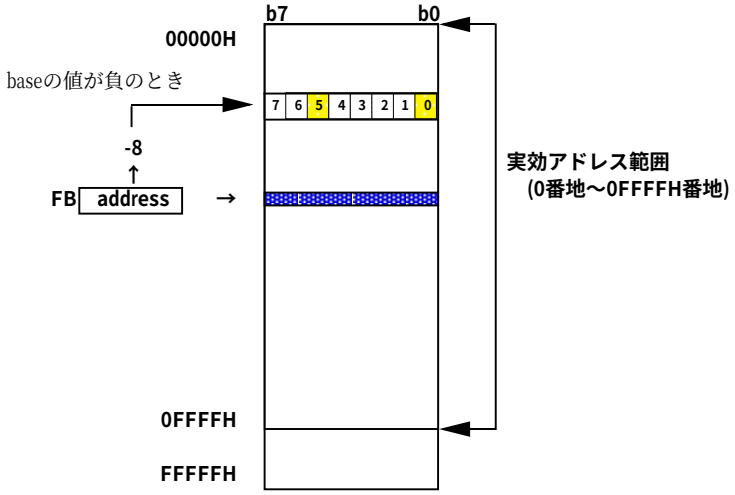


図 2.5.30　ビット命令 FB 相対アドレッシング

コラム　ビット数とアドレスの関係

ビット数をアドレスに置き換えるには、まずビット数を「バイト数＋ビット数」に変換します。変換は 1 バイトは 8 ビットなので、8 で除算します。変換を図 2.5.31 に示します。図のように 3 ビット右シフトを使って 1234H ビットは「246H バイト +4 ビット」になります。

主なアドレッシング計算例を図 2.5.32、図 2.5.33、図 2.5.34 に示します。

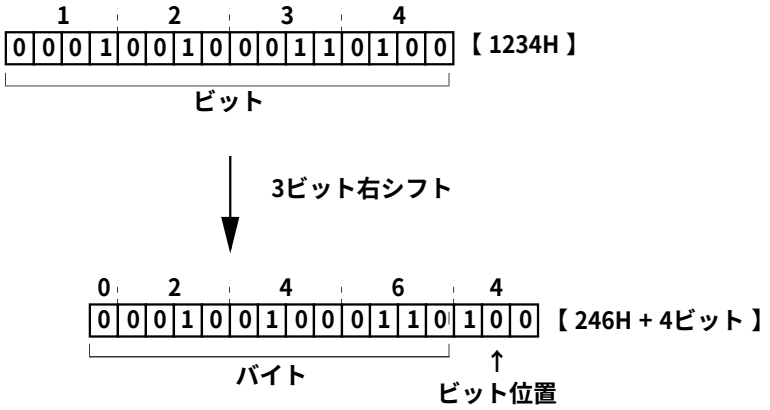


図 2.5.31　ビット数からアドレスへの変換

(1)アドレスレジスタ間接
(例)BCLR　[A0]

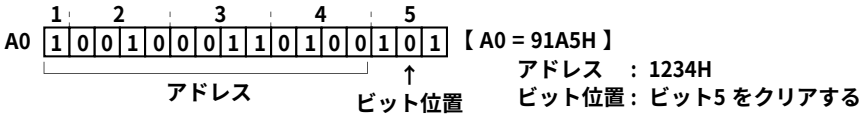


図 2.5.32　アドレスレジスタ間接アドレッシングのビット位置の計算

(2)アドレスレジスタ相対
(例)BCLR　5[A0]　　A0 はビット数です。dsp はアドレスなので 3 ビット左シフトして(バイト数で)計算します。

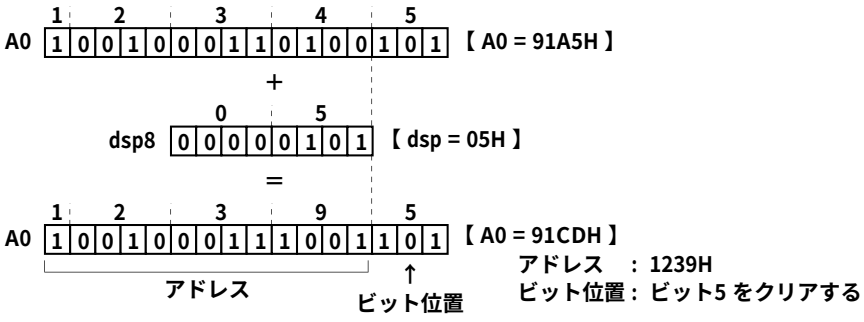


図 2.5.33　アドレスレジスタ相対アドレッシングのビット位置の計算

(3)SB 相対

(例)BCLR 5，0500H[SB]　　SB と base はアドレスなのでそのまま計算します。
bit はビット数なので 3 ビット右シフトして計算します。

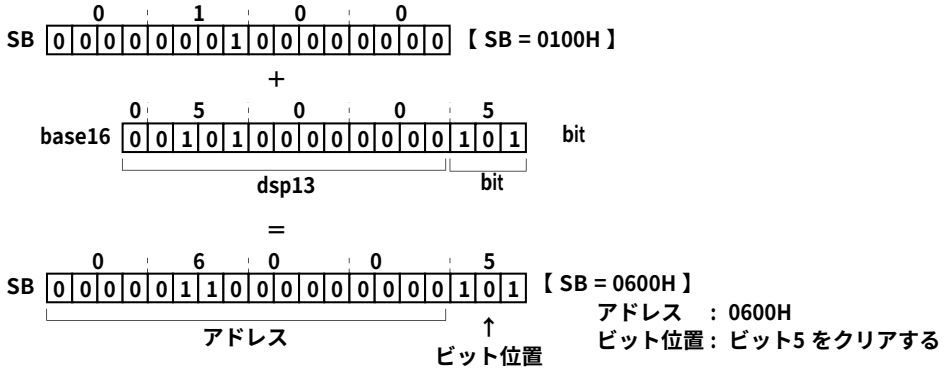


図 2.5.34　SB 相対アドレッシングのビット位置の計算

2.5.5 命令フォーマット

命令フォーマットには、ジェネリック、クイック、ショート、ゼロの4つの形式があります。アセンブラは、オペランドのバイト数が小さくなるように4つの形式から1つを選択しコードを生成する機能を持っています。生成コードを最適化する機能をアセンブラが持っているため、ユーザが指定する必要はありません。アセンブラが生成するコードの形式を指定したい場合のみ、フォーマット指定子をつけてください。

命令フォーマット

1. ジェネリック形式(:G)

オペコードには src と dest のアドレッシング情報も含まれます。

オペコード	srcコード	destコード
2バイト	0～3バイト	0～3バイト

2. クイック形式(:Q)

オペコードには動作および即値データと dest のアドレッシング情報も含まれます。ただし、オペコードに含まれる即値データは -7 ～ +8 または -8 ～ +7(命令によって異なります)で表現できる数値です。

オペコード	destコード
2バイト	0～2バイト

3. ショート形式(:S)

オペコードには src と dest のアドレッシング情報も含まれます。S 形式は一部のアドレッシングで使用します。

オペコード	srcコード	destコード
1バイト	0～2バイト	0～2バイト

4. ゼロ形式(:Z)

オペコードには動作および即値データと dest のアドレッシング情報も含まれます。　　ただし、即値データは 0 固定です。Z 形式は一部のアドレッシングで使用します。

オペコード	destコード
1バイト	0～2バイト

2.6 命令セット

M16C/60シリーズ、M16C/20シリーズの命令セットについて説明します。命令セットは機能別に一覧表としてまとめました。さらに命令セットの中から特長のある命令について詳しく説明します。

以下に一覧表の中で使用する記号と意味を示します。

記号	意味
src	処理結果を格納しないオペランド
dest	処理結果を格納するオペランド
label	アドレスを意味するオペランド
abs16	16 ビットの絶対値
abs20	20 ビットの絶対値
dsp:8	8 ビットの変位
dsp:16	16 ビットの変位
dsp:20	20 ビットの変位
#IMM	即値
.size	サイズ指定子(.B、.W)
.length	分岐距離指定子(.S、.B、.W、.A)
←	矢印の向きに転送
＋	加算
－	減算
×	乗算
÷	除算
∧	論理積
∨	論理輪
∀	排他的論理輪
¬	否定
| |	絶対値
EXT( )	( ) 内の符号拡張
U、I、O、B、S、Z、D、C	フラグ名
R0L、R0H、R1L、R1H	8 ビットのレジスタ名
R0、R1、R2、R3、A0、A1	16 ビットのレジスタ名
R2R0、R3R1、A1A0	32 ビットのレジスタ名
SB、FB、SP、PC	レジスタ名
MOV*Dir*、BM*Cnd*、J*Cnd*	*Dir*（方向）、*Cnd*（条件）のニーモニックは斜体で示す。
J*GEU/C*、J*EQ/Z*	J*GEU/C* は J*GEU* または J*C* と書き J*EQ/Z* は J*EQ* または J*Z* と書くことを示す。
"○"	（アドレッシング） 使用できる。
	（フラグ変化） 実行結果にしたがってフラグが変化する。
"－"	（フラグ変化） フラグは変化しない。

2.6.1 命令一覧

機能別に各ニーモニックの内容、アドレッシング、フラグの変化について示します。

転送

.sizeには.Wか.Bを記述します

ニーモニック		説明
MOV.size	src,dest	src を dest に転送または、即値を dest にセット
MOVA	src,dest	src のアドレスを dest に転送
MOV*HH*	src,dest	src の上位 4 ビットを dest の上位 4 ビットに転送
MOV*HL*	src,dest	src の上位 4 ビットを dest の下位 4 ビットに転送
MOV*LH*	src,dest	src の下位 4 ビットを dest の上位 4 ビットに転送
MOV*LL*	src,dest	src の下位 4 ビットを dest の下位 4 ビットに転送
POP.size	dest	スタック領域から値を復帰
POPM	dest	複数のレジスタの値を一括してスタック領域から復帰
PUSH.size	src	スタック領域へレジスタ / メモリ / 即値を退避
PUSHA	src	src のアドレスをスタック領域へ退避
PUSHM	src	複数のレジスタをスタック領域へ退避
LDE.size	src,dest	拡張データ領域にある src を転送
STE.size	src,dest	src を拡張データ領域へ転送
STNZ	src,dest	Z フラグが "0" のとき src を転送
STZ	src,dest	Z フラグが "1" のとき src を転送
STZX	src1,src2,dest	Z フラグが "1" のとき src1 を転送し、"0" のとき src2 を転送
XCHG.size	src,dest	src と dest を交換

＊a　src または dest に R0L レジスタを選択します。
＊b　R0L、R0H、R1L、R1H から選択できます。
＊c　即値は 8 ビットです。
＊d　R0L か R0H を選択します。
＊e　dsp:8[SB]か dsp:8[FB]を選択します。

アドレッシング											フラグ変化							
オペランド	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
src	○	○	○	○	○						－	－	－	－	○	○	－	－
dest		○	○	○	○													
src		○			○						－	－	－	－	－	－	－	－
dest			○															
src			R0L＊a								－	－	－	－	－	－	－	－
dest		○	○＊b	○	○													
src		○	○＊b	○	○													
dest			R0L＊a															
dest		○	○	○	○						－	－	－	－	－	－	－	－
dest			○							○	－	－	－	－	－	－	－	－
src	○	○	○	○	○						－	－	－	－	－	－	－	－
src		○			○						－	－	－	－	－	－	－	－
src			○							○	－	－	－	－	－	－	－	－
src						○		○	dsp:20[A0]		－	－	－	－	○	○	－	－
dest		○	○	○	○													
src		○	○	○	○						－	－	－	－	○	○	－	－
dest						○		○	dsp:20[A0]									
src	○＊c										－	－	－	－	－	－	－	－
dest		○	○＊d		○＊e													
src	○＊c										－	－	－	－	－	－	－	－
dest		○	○＊d		○＊e													
src1,src2	○＊c										－	－	－	－	－	－	－	－
dest		○	○＊d		○＊e													
src			○								－	－	－	－	－	－	－	－
dest		○	○	○	○													

ビット処理

ニーモニック		説明	
BAND	src	C フラグ ← src ∧ C フラグ	;ビット論理積
BCLR	dest	dest ← 0	;ビットクリア
BM*GEU/C*	dest	C=1 ならば dest ← 1、 それ以外は dest ← 0	;条件ビット転送
BM*LTU/NC*	dest	C=0 ならば dest ← 1、 それ以外は dest ← 0	
BM*EQ/Z*	dest	Z=1 ならば dest ← 1、 それ以外は dest ← 0	
BM*NE/NZ*	dest	Z=0 ならば dest ← 1、 それ以外は dest ← 0	
BM*GTU*	dest	C∧$\overline{Z}$=1 ならば dest ← 1、 それ以外は dest ← 0	
BM*LEU*	dest	C∧$\overline{Z}$=0 ならば dest ← 1、 それ以外は dest ← 0	
BM*PZ*	dest	S=0 ならば dest ← 1、 それ以外は dest ← 0	
BM*N*	dest	S=1 ならば dest ← 1、 それ以外は dest ← 0	
BM*GE*	dest	S∀O=0 ならば dest ← 1、 それ以外は dest ← 0	
BM*LE*	dest	(S∀O)∨Z=1 ならば dest ← 1、 それ以外は dest ← 0	
BM*GT*	dest	(S∀O)∨Z=0 ならば dest ← 1、 それ以外は dest ← 0	
BM*LT*	dest	S∀O=1 ならば dest ← 1、 それ以外は dest ← 0	
BM*O*	dest	O=1 ならば dest ← 1、 それ以外は dest ← 0	
BM*NO*	dest	O=0 ならば dest ← 1、 それ以外は dest ← 0	
BNAND	src	C フラグ ← $\overline{src}$ ∧ C フラグ	;反転ビット論理積
BNOR	src	C フラグ ← $\overline{src}$ ∨ C フラグ	;反転ビット論理和
BNOT	dest	dest を反転し、dest に格納	;ビット反転
BNTST	src	Z フラグ ← $\overline{src}$、 C フラグ ← $\overline{src}$	;反転ビットテスト
BNXOR	src	C フラグ ← $\overline{src}$ ∀ C フラグ	;反転ビット排他的論理和
BOR	src	C フラグ ← src ∨ C フラグ	;ビット論理和
BSET	dest	dest ← 1	;ビットセット
BTST	src	Z フラグ ← $\overline{src}$、 C フラグ ← src	;ビットテスト
BTSTC	dest	Z フラグ ← $\overline{dest}$、 C フラグ ← dest、 dest ← 0	;ビットテスト＆クリア
BTSTS	dest	Z フラグ ← $\overline{dest}$、 C フラグ ← dest、 dest ← 1	;ビットテスト＆セット
BXOR	src	C フラグ ← src ∀ C フラグ	;ビット排他的論理輪

オペランド	アドレッシング					フラグ変化							
	ビット命令												
	絶対	レジスタ直接	レジスタ間接	レジスタ相対	フラグ直接	U	I	O	B	S	Z	D	C
src	○	○	○	○		—	—	—	—	—	—	—	○
dest	○	○	○	○		—	—	—	—	—	—	—	—
dest	○	○	○	○	○	—	—	—	—	—	—	—	○*f
src	○	○	○	○		—	—	—	—	—	—	—	○
src	○	○	○	○		—	—	—	—	—	—	—	○
dest	○	○	○	○		—	—	—	—	—	—	—	—
src	○	○	○	○		—	—	—	—	—	○	—	○
src	○	○	○	○		—	—	—	—	—	—	—	○
src	○	○	○	○		—	—	—	—	—	—	—	○
dest	○	○	○	○		—	—	—	—	—	—	—	—
src	○	○	○	○		—	—	—	—	—	○	—	○
dest	○	○	○	○		—	—	—	—	—	○	—	○
dest	○	○	○	○		—	—	—	—	—	○	—	○
src	○	○	○	○		—	—	—	—	—	—	—	○

＊f　dest に C フラグを指定したとき、変化します。

算術

.sizeには.Wか.Bを記述します

ニーモニック		説明	
ABS.size	dest	dest ← |dest|	;dest の絶対値
ADC.size	src,dest	dest ← src + dest + C フラグ	;キャリー付き 16 進加算
ADCF.size	dest	dest ← dest + C フラグ	;キャリーフラグの加算
ADD.size	src,dest	dest ← src + dest	;キャリーなし 16 進加算
CMP.size	src,dest	dest - src	;比較、結果はフラグで判断
DADC.size	src,dest	dest ← src + dest + C フラグ	;キャリー付き 10 進加算
DADD.size	src,dest	dest ← src + dest	;キャリーなし 10 進加算
DEC.size	dest	dest ← dest - 1	;デクリメント
DIV.size	src	R0(商),R2(剰余) ← R2R0 ÷ src	;符号付き除算
DIVU.size	src	R0(商),R2(剰余) ← R2R0 ÷ src	;符号なし除算
DIVX.size	src	R0(商),R2(剰余) ← R2R0 ÷ src	;符号付き除算
DSBB.size	src,dest	dest ← dest - src - $\overline{\text{C フラグ}}$	;ボロー付き 10 進減算
DSUB.size	src,dest	dest ← dest - src	;ボローなし 10 進減算
EXTS.size	dest	dest ← EXT(dest)	;dest の符号拡張
INC.size	dest	dest ← dest + 1	;インクリメント
MUL.size	src,dest	dest ← dest × src	;符号付き乗算

オペランド	アドレッシング										フラグ変化							
	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
src		○	○	○	○						−	−	○	−	○	○	−	○
src	○	○	○	○	○						−	−	○	−	○	○	−	○
dest		○	○	○	○													
dest		○	○	○	○						−	−	○	−	○	○	−	○
src	○	○	○	○	○						−	−	○	−	○	○	−	○
dest		○	○	○	○					SP								
src	○	○	○	○	○						−	−	○	−	○	○	−	○
dest		○	○	○	○													
src	○		○*g								−	−	−	−	○	○	−	○
dest			○*g															
src	○		○*g								−	−	−	−	○	○	−	○
dest			○*g															
dest		○	○*h		○*i						−	−	−	−	○	○	−	−
src	○	○	○	○	○						−	−	○	−	−	−	−	−
dest	○	○	○	○	○						−	−	○	−	−	−	−	−
src	○	○	○	○	○						−	−	○	−	−	−	−	−
src	○		○*g								−	−	−	−	○	○	−	○
dest			○*g															
src	○		○*g								−	−	−	−	○	○	−	○
dest			○*g															
dest		○	○*j	○	○						−	−	−	−	○	○	−	−
dest		○	○*h		○*i						−	−	−	−	○	○	−	−
src	○	○	○	○	○						−	−	−	−	−	−	−	−
dest		○	○	○	○													

＊g　src は R0H、R1 から、dest は R0L、R0 から選択します。
＊h　R0L、R0H、A0、A1 から選択します。
＊i　dsp:8[SB]か dsp:8[FB]を選択します。
＊j　R0L、R0、R1L から選択します。

.sizeには.Wか.Bを記述します

ニーモニック	説明
MULU.size src,dest	dest ← dest × src ;符号なし乗算
NEG.size dest	dest ← 0 - dest ;2 の補数
RMPA.size	R2R0 ← A0 を被乗数番地、A1 を乗数番地、R3 を回数とする積和演算 ;積和演算
SBB.size src,dest	dest ← dest - src - $\overline{\text{C フラグ}}$;ボロー付き減算
SUB.size src,dest	dest ← dest - src ;ボローなし減算

アドレッシング											フラグ変化							
オペランド	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
src	○	○	○	○	○						—	—	—	—	—	—	—	—
dest		○	○	○	○													
dest		○	○	○	○						—	—	○	—	○	○	—	○
-											—	—	○	—	—	—	—	—
src	○	○	○	○	○						—	—	○	—	○	○	—	○
dest		○	○	○	○													
src	○	○	○	○	○						—	—	○	—	○	○	—	○
dest		○	○	○	○													

論理

.sizeには.Wか.Bを記述します

ニーモニック		説明	
AND.size	src,dest	dest ← src ∧ dest	;論理積
NOT.size	dest	dest ← $\overline{\text{dest}}$	;全ビット反転
OR.size	src,dest	dest ← src ∨ dest	;論理和
TST.size	src,dest	src ∧ dest	;テスト
XOR.size	src,dest	dest ← dest ∀ src	;排他的論理和

シフト

.sizeには.Wか.Bを記述します

ニーモニック		説明
ROLC.size	dest	dest を C フラグを含めて 1 ビット左へ回転
RORC.size	dest	dest を C フラグを含めて 1 ビット右へ回転
ROT.size	src,dest	dest を src で指定したビット数分回転
SHA.size	src,dest	dest を src で示すビット数分算術シフト
SHL.size	src,dest	dest を src で示すビット数分論理シフト

オペランド	アドレッシング										フラグ変化							
	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
src	○	○	○	○	○						—	—	—	—	○	○	—	—
dest		○	○	○	○													
dest		○	○	○	○						—	—	—	—	○	○	—	—
src	○	○	○	○	○						—	—	—	—	○	○	—	—
dest		○	○	○	○													
src	○	○	○	○	○						—	—	—	—	○	○	—	—
dest		○	○	○	○													
src	○	○	○	○	○						—	—	—	—	○	○	—	—
dest		○	○	○	○													

オペランド	アドレッシング										フラグ変化							
	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
dest		○	○	○	○						—	—	—	—	○	○	—	○
dest		○	○	○	○						—	—	—	—	○	○	—	○
src	○*k		○								—	—	—	—	○	○	—	○
dest		○	○	○	○													
src	○*k		R1H								—	—	○	—	○	○	—	○
dest		○	○	○	○					○*l								
src	○*k		R1H								—	—	—	—	○	○	—	○
dest		○	○	○	○					○*l								

＊k　即値の取りうる範囲は -8 ≦ #IMM ≦ +8 です。ただし 0 は設定できません。
＊l　R2R0 か R3R1 を選択します。

ジャンプ

.sizeには.Wか.Bを記述します

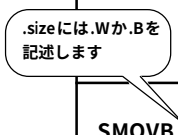
.lengthには.Aか.Wを記述します

ニーモニック	説明
ADJNZ.size src,dest,label	dest ← dest + src の結果が 0 以外であれば label へ分岐;加算 & 条件分岐
SBJNZ.size src,dest,label	dest ← dest - src の結果が 0 以外であれば label へ分岐;減算 & 条件分岐
JGEU/C label	C=1 ならば label へ分岐、それ以外は次の命令を実行 ;条件分岐
JLTU/NC label	C=0 ならば label へ分岐、それ以外は次の命令を実行
JEQ/Z label	Z=1 ならば label へ分岐、それ以外は次の命令を実行
JNE/NZ label	Z=0 ならば label へ分岐、それ以外は次の命令を実行
JGTU label	C∧Z=1 ならば label へ分岐、それ以外は次の命令を実行
JLEU label	C∧Z=0 ならば label へ分岐、それ以外は次の命令を実行
JPZ label	S=0 ならば label へ分岐、それ以外は次の命令を実行
JN label	S=1 ならば label へ分岐、それ以外は次の命令を実行
JGE label	S∀O=1 ならば label へ分岐、それ以外は次の命令を実行
JLE label	(S∀O)∨Z=1 ならば label へ分岐、それ以外は次の命令を実行
JGT label	(S∀O)∨Z=0 ならば label へ分岐、それ以外は次の命令を実行
JLT label	S∀O=1 ならば label へ分岐、それ以外は次の命令を実行
JO label	O=1 ならば label へ分岐、それ以外は次の命令を実行
JNO label	O=0 ならば label へ分岐、それ以外は次の命令を実行
JMP label	label へ分岐 ;無条件分岐
JMPI.length src	src が示す番地へ分岐 ;間接分岐
JMPS src	スペシャルページ分岐
JSR label	サブルーチン呼び出し
JSR.length src	間接サブルーチン呼び出し
JSRS src	スペシャルページサブルーチン呼び出し
RTS	サブルーチンからの復帰

アドレッシング											フラグ変化							
オペランド	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
src	○*m																	
dest		○	○	○	○						—	—	—	—	—	—	—	—
label									label *p									
src	○*n																	
dest		○	○	○	○						—	—	—	—	—	—	—	—
label									label *p									
label									label *q		—	—	—	—	—	—	—	—
label						○			label *r		—	—	—	—	—	—	—	—
src		○	○	○	○				dsp:20[A0]		—	—	—	—	—	—	—	—
src	○*o										—	—	—	—	—	—	—	—
label						○			label *r		—	—	—	—	—	—	—	—
src		○	○	○	○				dsp:20[A0]		—	—	—	—	—	—	—	—
dest	○*o										—	—	—	—	—	—	—	—
-											—	—	—	—	—	—	—	—

＊m　即値の範囲は -8 ≦ #IMM ≦ +7 です。
＊n　即値の範囲は -7 ≦ #IMM ≦ +8 です。
＊o　即値は 8 ビットです。
＊p　label の範囲は PC-126 ≦ label ≦ PC+129 です。
＊q　条件が LE,O,GE,GT,NO,LT の場合、label の範囲は、-126 ≦ label ≦ PC+129 です。
　　それ以外は -127 ≦ label ≦ PC+128 です。
＊r　label の範囲は PC-32767 ≦ label ≦ PC+32768 です。

ストリング



ニーモニック	説明
SMOVB.size	R1H,A0 を転送元番地、A1 を転送先番地、R3 を転送回数として、アドレスの減算方向へストリング転送
SMOVF.size	R1H,A0 を転送元番地、A1 を転送先番地、R3 を転送回数として、アドレスの加算方向へストリング転送
SSTR.size	R0 を転送データ、A1 を転送先番地、R3 を転送回数として、アドレスの加算方向へストリングストア

オペランド	アドレッシング										フラグ変化							
	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ 直接	U	I	O	B	S	Z	D	C
-											—	—	—	—	—	—	—	—
-	注意: ストリングにはあてはまるアドレッシングがありません。										—	—	—	—	—	—	—	—
-											—	—	—	—	—	—	—	—

その他

ニーモニック		説明
BRK		BRK 割り込みの発生
ENTER	src	スタックフレームの構築
EXITD		スタックフレームを解放とサブルーチンから復帰
FCLR	dest	dest のフラグのクリア
FSET	dest	dest のフラグのセット
INT	src	ソフトウェア割り込みの発生
INTO		O フラグが "1" のとき、オーバフロー割り込みの発生
LDC	src,dest	src の専用レジスタへの転送
LDCTX	abs16,abs20	タスクのコンテキストをスタックから復帰
LDINTB	src	src を INTB に転送
LDIPL	src	src を IPL に転送
NOP		ノーオペレーション
POPC	dest	スタック領域から専用レジスタを復帰
PUSHC	src	スタック領域へ専用レジスタを退避
REIT		割込ルーチンから戻る ;割り込みからの復帰
STC	src,dest	専用レジスタから dest に転送
STCTX	abs16,abs20	タスクのコンテキストをスタックへ退避
UND		未定義命令割り込みの発生
WAIT		プログラムの停止。割り込みまたはリセットによりプログラムの再開可能。

オペランド	アドレッシング										フラグ変化							
	一般命令					特定命令												
	即値	16 ビット絶対	レジスタ直接	レジスタ間接	レジスタ相対	20 ビット絶対	32 ビット レジスタ直接	32 ビット レジスタ間接	20 ビット レジスタ相対	専用レジスタ直接	U	I	O	B	S	Z	D	C
-											—	—	—	—	—	—	—	—
src	○*s										—	—	—	—	—	—	—	—
-											—	—	—	—	—	—	—	—
dest										○	選択したフラグが "0" になります							
dest										○	選択したフラグが "1" になります							
src	○*t										○	○	—	—	—	—	○	—
-											○	○	—	—	—	—	○	—
src	○	○	○	○	○						dest が FLG のときだけ変化します							
dest										○*w								
dest		○				○					—	—	—	—	—	—	—	—
src	○*u										—	—	—	—	—	—	—	—
src	○*v										—	—	—	—	—	—	—	—
-											—	—	—	—	—	—	—	—
dest										○*w	dest が FLG のときだけ変化します							
src										○*w	—	—	—	—	—	—	—	—
-											割り込み要求が受け付けられる前の FLG の状態に戻ります							
src										○	—	—	—	—	—	—	—	—
dest		○	○	○	○													
src		○				○					—	—	—	—	—	—	—	—
-			○								○	○	—	—	—	—	○	—
-											—	—	—	—	—	—	—	—

＊s　即値は 8 ビットで指定できます。
＊t　即値の範囲は 0 ≦ #IMM ≦ 63 です。
＊u　即値は 20 ビットで指定できます。
＊v　即値の範囲は 0 ≦ #IMM ≦ 7 です。
＊w　PC レジスタを除く専用レジスタを選択できます。

2.6.2　転送命令とストリング命令

転送は通常バイトまたはワード単位で行います。転送命令は14命令用意されています。その中には、4ビットだけを転送する4ビット転送命令や、条件分岐を組み合わせた条件ストア命令、データをまとめて転送するストリング命令があります。

この項ではデータ転送に関する命令の中から上記の特長のある命令3種類について説明します。

4ビット転送命令

4ビット転送命令は、8ビットのレジスタ / メモリの上位4ビットまたは下位4ビットを転送する命令です。4ビット転送はアンパックド BCD コードの作成、4ビット単位の I/O ポート入出力に利用できます。

転送する4ビットが、上位か下位かによって *Dir* に入るニーモニックが変わります。

この4ビット転送命令では src または dest のどちらか一方を R0L にしてください。

表 2.6.1　4ビット転送命令

ニーモニック	記述形式		説明
MOVDir			転送
	MOVHH	src,dest	上位4ビット:src → 上位4ビット:dest
	MOVHL	src,dest	上位4ビット:src → 下位4ビット:dest
	MOVLH	src,dest	下位4ビット:src → 上位4ビット:dest
	MOVLL	src,dest	下位4ビット:src → 下位4ビット:dest

(注)srcかdestのどちらか一方は必ずR0L

条件ストア命令

条件ストア命令は、Z フラグの状態が条件となる条件付き転送命令です。この命令は、条件判断と転送を 1 命令で行うことができます。

条件ストア命令は、STZ、STNZ、STZX の 3 種類です。動作例を図 2.6.1 に示します。

表 2.6.2　条件ストア命令

ニーモニック	記述形式	説明
STZ	STZ　src,dest	Zフラグ=1のとき srcをdestに転送する
STNZ	STNZ　src,dest	Zフラグ=0のとき srcをdestに転送する
STZX	STZX　src1,src2,dest	Zフラグ=1のとき src1をdestに転送する Zフラグ=0のとき src2をdestに転送する

(注)src,src1,src2は、#IMM8(8ビット即値)だけ

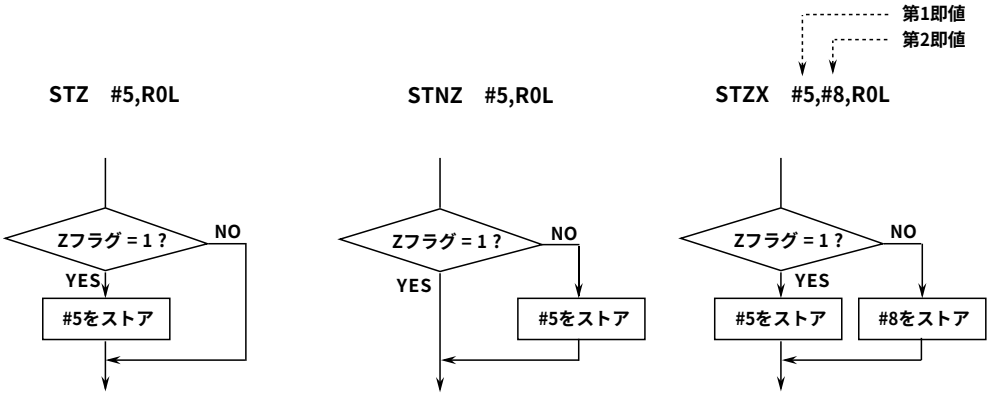


図 2.6.1　条件ストア命令の動作例

ストリング命令

ストリング命令はデータをまとめて転送する命令です。ブロック転送や RAM クリアに利用します。

図2.6.2に示すように、各レジスタに転送元アドレス、転送先アドレス、転送回数をセットして命令を実行します。転送はバイトまたはワード単位で行います。ストリング命令の動作例を図 2.6.3 に示します。

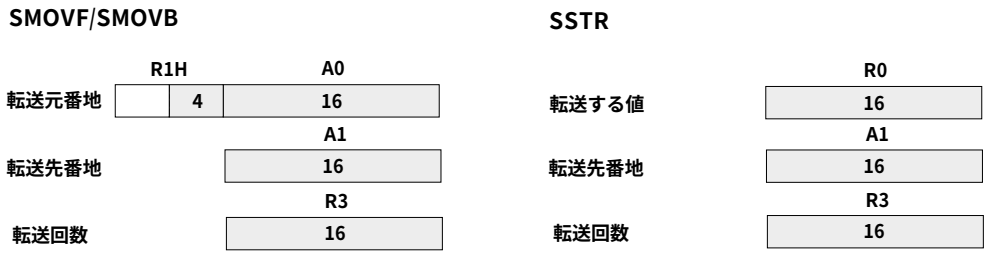


図 2.6.2　ストリング命令のレジスタ設定

表 2.6.3　ストリング命令

ニーモニック	記述形式	説明
SMOVF	SMOVF .B SMOVF .W	アドレスの加算方向へストリング転送する
SMOVB	SMOVB .B SMOVB .W	アドレスの減算方向へストリング転送する
SSTR	SSTR .B SSTR .W	アドレスの加算方向へストリングストアする

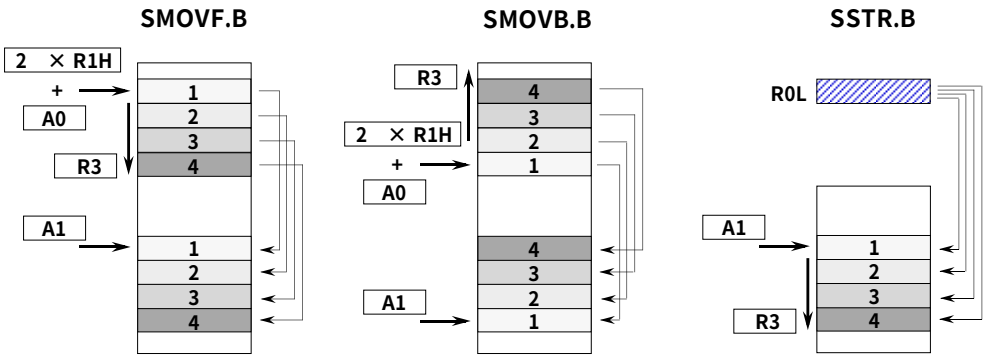


図 2.6.3　ストリング命令の動作例

2.6.3 演算命令

演算命令は 31 命令用意されています。この項では特長のある演算命令を説明します。

乗算命令

乗算命令には符号付き乗算命令と符号なし乗算命令があります。2 つの命令はサイズ指定ができます。.B を指定すると(8 ビット)×(8 ビット)=(16 ビット)で演算し、.W を指定すると(16 ビット)×(16 ビット)=(32 ビット)で演算します。

.B を指定した場合には、src と dest の両方にアドレスレジスタを使うことはできません。また、乗算命令ではフラグは変化しません。乗算命令の動作例を図 2.6.4 に示します。

表 2.6.4 乗算命令

ニーモニック	記述形式	説明
MUL	MUL.B src,dest MUL.W src,dest	符号付き乗算命令 dest ← src × dest
MULU	MULU.B src,dest MULU.W src,dest	符号なし乗算命令 dest ← src × dest

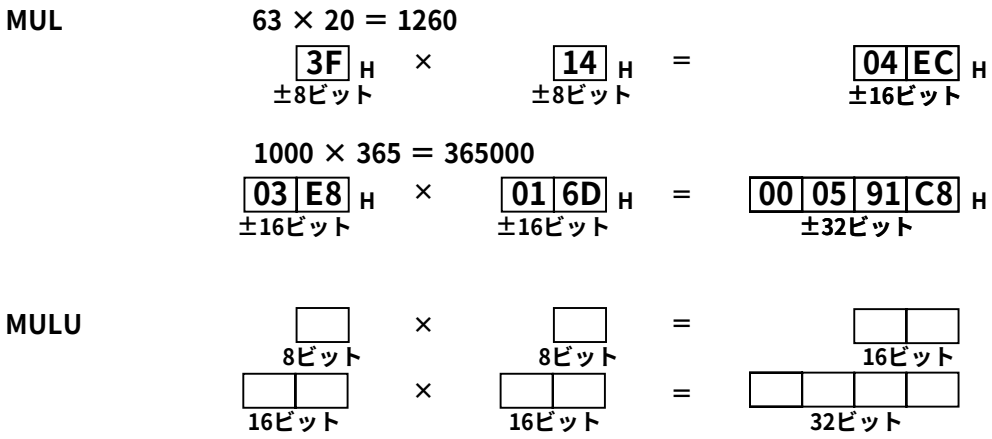


図 2.6.4 乗算命令の動作例

除算命令

除算命令には符号付き除算命令が 2 つと符号なし除算命令が 1 つがあります。3 つの命令はサイズ指定ができます。.B を指定すると(16 ビット)÷(8 ビット)=(8 ビット)...(余り 8 ビット)で演算し、.W を指定すると(32 ビット)÷(16 ビット)=(16 ビット)...(余り 16 ビット)で演算します。

また、除算命令では O フラグだけ変化します。除算命令の動作例を図 2.6.5 に示します。

表 2.6.5　除算命令

ニーモニック	記述形式		説明
DIV	DIV.B DIV.W	src src	符号付き除算命令 余りの符号は被除数の符号に一致
DIVX	DIVX.B DIVX.W	src src	符号付き除算命令 余りの符号は除数の符号に一致
DIVU	DIVU.B DIVU.W	src src	符号なし除算命令

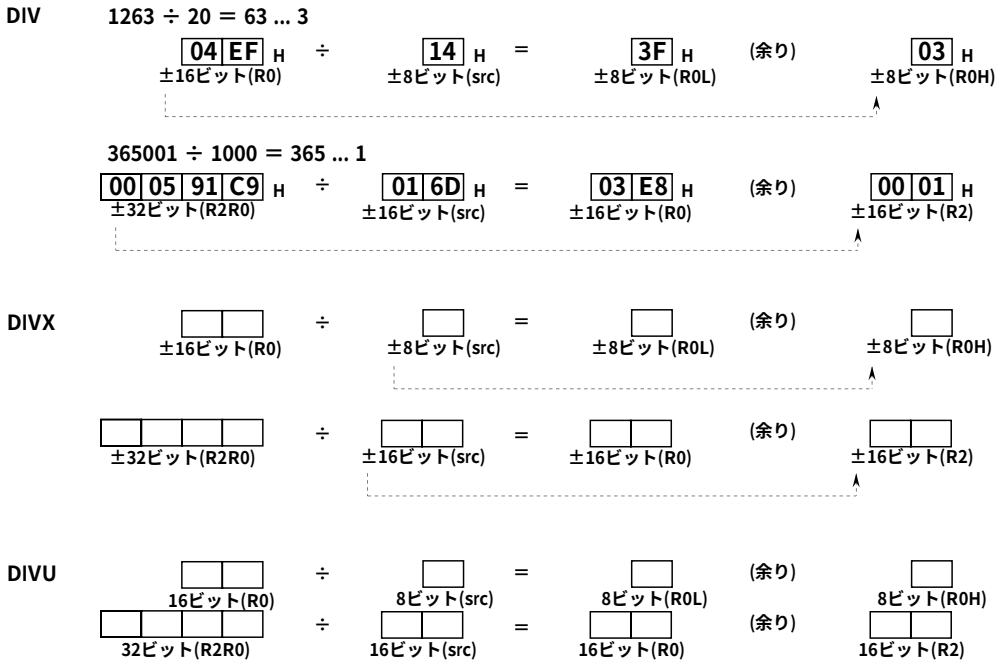


図 2.6.5　除算命令の動作例

DIV 命令と DIVX 命令の違いについて

DIV と DIVX は共に符号付きの除算命令ですが、2 つの命令の違いは余りの符号です。
表 2.6.6 に示すように、DIV 命令では余りの符号は被除数と同じで、DIVX 命令では除数と同じになります。

表 2.6.6　DIV 命令と DIVX 命令の違い

DIV	33 ÷ 4 = 8 ... 1	余りの符号は被除数と同じ
	33 ÷ (-4) = -8 ... 1	
	-33 ÷ 4 = -8 ... (-1)	
DIVX	33 ÷ 4 = 8 ... 1	余りの符号は除数と同じ
	33 ÷ (-4) = -9 ... (-3)	
	-33 ÷ 4 = -9 ... 3	

10 進加算命令

10 進加算命令にはキャリーなし 10 進加算命令とキャリー付き 10 進加算命令の 2 つがあります。
10 進加算命令を実行すると S フラグ、Z フラグ、C フラグが変化します。動作例を図 2.6.6 に示します。

表 2.6.7　10 進加算命令

ニーモニック	記述形式	説明
DADD	DADD .B src,dest DADD .W src,dest	10進でキャリーを含まずに加算
DADC	DADC .B src,dest DADC .W src,dest	10進でキャリーを含んで加算

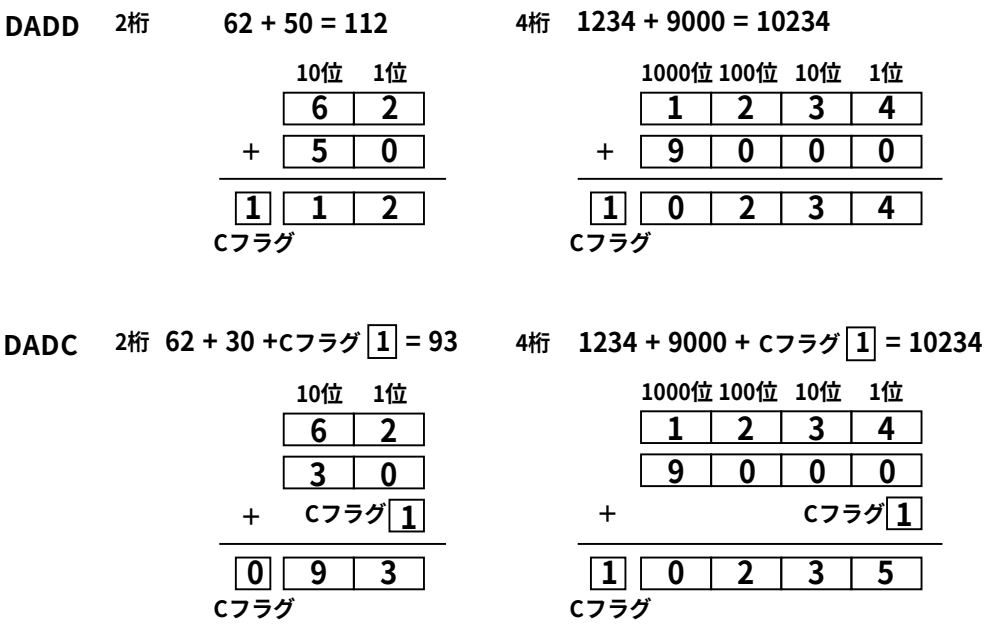


図 2.6.6　10 進加算命令の動作例

10 進減算命令

10 進減算命令にはボローなし 10 進減算命令とボロー付き 10 進減算命令の 2 つがあります。
10 進減算命令を実行すると S フラグ、Z フラグ、C フラグが変化します。動作例を図 2.6.7 に示します。

表 2.6.8　10 進減算命令

ニーモニック	記述形式	説明
DSUB	DSUB .B src,dest DSUB .W src,dest	ボロー含まずに10進で減算
DSBB	DSBB .B src,dest DSBB .W src,dest	ボローを含んで10進で減算

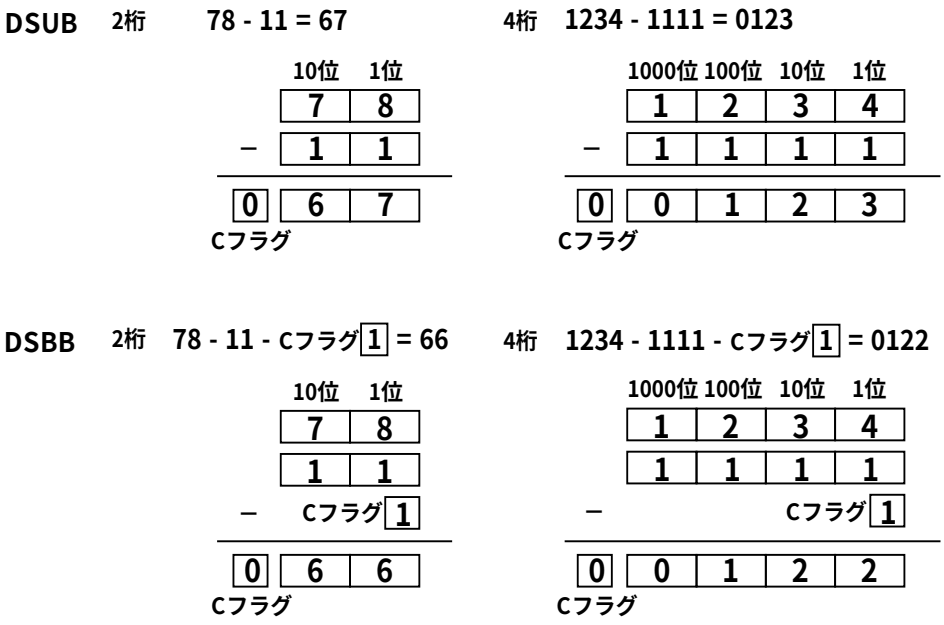


図 2.6.7　10 進減算命令の動作例

加算(減算)& 条件分岐命令

加算(減算)&条件分岐命令は、繰り返し処理の終了判断に便利な命令です。この命令で加算/減算する値は4ビット即値に限られます。ADJNZ 命令の場合は -8 ～ +7、SBJNZ 命令の場合は -7 ～ +8 になります。分岐できる範囲は、ADJNZ/SBJNZ 命令の先頭番地から -126 ～ +129 の領域です。加算(減算)& 条件分岐命令の動作例を図 2.6.8 に示します。

表 2.6.9　加算(減算)& 条件分岐命令

ニーモニック	記述形式	説明
ADJNZ	ADJNZ.B　#IMM,dest,label ADJNZ.W　#IMM,dest,label	destに即値を加算。 加算結果が0以外のときlabelへ分岐
SBJNZ	SBJNZ.B　#IMM,dest,label SBJNZ.W　#IMM,dest,label	destから即値を減算。 減算結果が0以外のときlabelへ分岐

(注1)#IMM: 4ビット即値だけ (ADJNZの場合:-8～+7,SBJNZの場合:-7～+8)
(注2)分岐範囲 : PC相対アドレスで、ADJNZ/SBJNZ命令の先頭番地から-126～+129

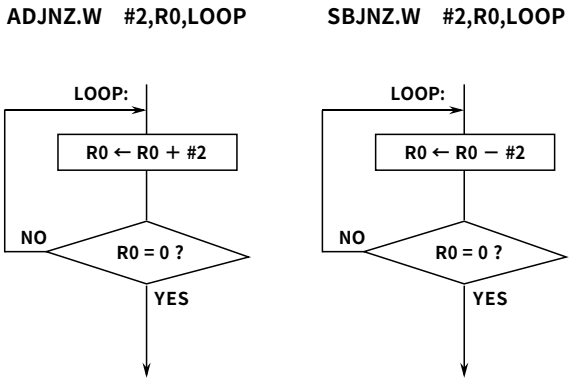


図 2.6.8　加算(減算)& 条件分岐命令の動作例

積和演算命令

積和演算命令は、積和演算を行い、演算中にオーバフローが発生したらオーバフロー割り込みを発生します。図 2.6.9 に示すように、被乗数アドレス、乗数アドレス、積和回数は、各レジスタに設定します。積和演算命令の動作例を図 2.6.10 に示します。

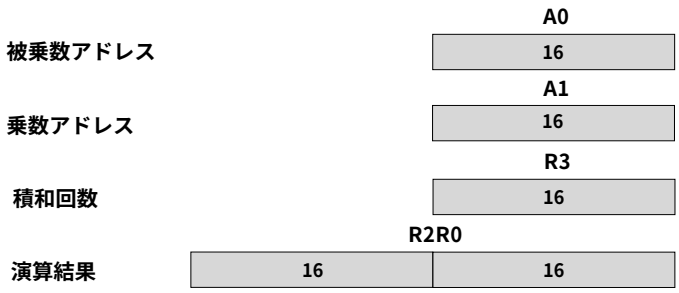


図 2.6.9　積和演算命令のレジスタ設定

表 2.6.10　積和演算命令

ニーモニック	記述形式		説明
RMPA	RMPA	.B .W	A0を被乗数アドレス、A1を乗数アドレス、 R3を回数、とする積和演算

(注1)演算中にオーバフローが発生した場合、オーバフローフラグ(Oフラグ)を1にセットし演算を終了する。
(注2)演算中に割り込み要求があった場合は、演算途中の加算終了後積和回数を減算し、割り込みを受け付ける。

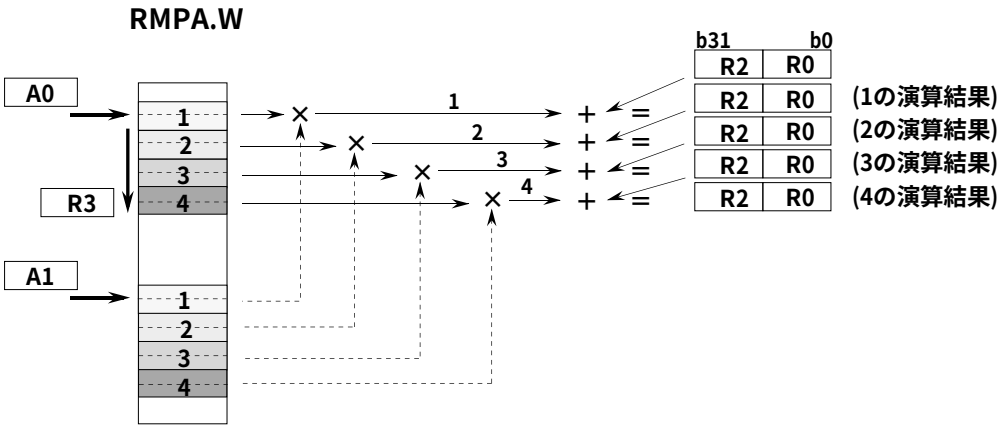


図 2.6.10　積和演算命令の動作例

2.6.4 符号拡張命令

符号拡張命令は、符号ビットを拡張するビットに代入しビット長を長くする命令です。
この項では符号拡張命令について説明します。

符号拡張命令

符号拡張命令は、8 ビット符号拡張と 16 ビット符号拡張を行います。

サイズ指定子に .W を指定すると 16 ビットから 32 ビットへの符号拡張となりますが、この場合は R0 レジスタを使用してください。符号拡張命令の動作例を図 2.6.11 に示します。

表 2.6.11　符号拡張命令

ニーモニック	記述形式	説明
EXTS	EXTS.B dest EXTS.W R0	destを8ビットから16ビットへまたは16ビット(R0)から32ビット(R2R0)へ符号拡張する。

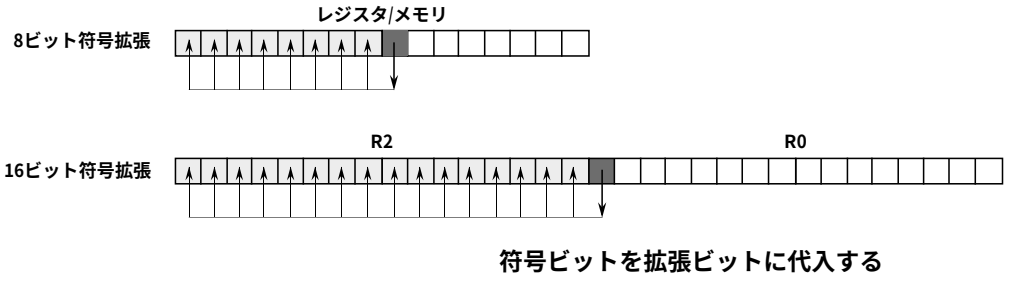


図 2.6.11　符号拡張命令の動作例

2.6.5 ビット命令

この項ではビット命令について説明します。

ビット論理演算命令

ビット論理演算命令は、レジスタ / メモリの指定ビットと、C フラグを論理演算し、結果を C フラグに格納する命令です。ビット論理演算命令の動作例を図 2.6.12 に示します。

表 2.6.12　ビット論理演算命令

ニーモニック	記述形式	説明
BAND	BAND src	C ← src ∧ C ; Cとsrcの論理積
BNAND	BNAND src	C ← $\overline{src}$ ∧ C ; Cと$\overline{src}$の論理積
BOR	BOR src	C ← src ∨ C ; Cとsrcの論理和
BNOR	BNOR src	C ← $\overline{src}$ ∨ C ; Cと$\overline{src}$の論理和
BXOR	BXOR src	C ← src ∀ C ; Cとsrcの排他的論理和
BNXOR	BNXOR src	C ← $\overline{src}$ ∀ C ; Cと$\overline{src}$の排他的論理和

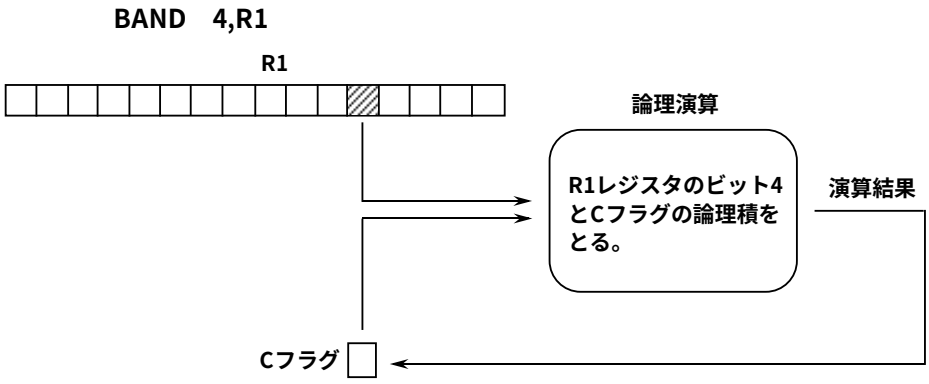


図 2.6.12　ビット論理演算命令の動作例

条件ビット転送命令

条件ビット転送命令は、条件付きのビット転送命令です。条件が真ならば“1”を転送し、偽ならば“0”を転送します。真偽判定はすべてフラグで行いますので、この命令の前にはフラグが変化する演算命令が必要です。条件ビット転送命令の動作例を図 2.6.13 に示します。

表 2.6.13　条件ビット転送命令

ニーモニック	記述形式	説明
BMCnd	BMCnd　dest BMCnd　C	条件が真ならば“1”を転送し、 条件が偽ならば“0”を転送する。

Cnd	真偽判定条件(14種類)	
GEU/C	C = 1	等しいまたは大きい/キャリーフラグが“1”
GTU	C = 1 & Z = 0	符号なしで大きい
EQ/Z	Z = 1	等しい/ゼロフラグが“1”
N	S = 1	負
LE	(Z = 1) ¦ (S = 1 & O = 0) ¦ (S = 0 & O = 1)	等しいまたは符号付きで小さい
O	O = 1	オーバフローフラグが“1”
GE	(S = 1 & O = 1) ¦ (S = 0 & O = 0)	等しいまたは符号付きで大きい
LTU/NC	C = 0	小さい/キャリーフラグが“0”
LEU	C = 0 ¦ Z = 1	等しいまたは小さい
NE/NZ	Z = 0	等しくない/ゼロフラフが“0”
PZ	S = 0	正またはゼロ
GT	(S = 1 & O = 1 & Z = 0) ¦ (S = 0 & O = 0 & Z = 0)	符号付きで大きい
NO	O = 0	オーバフローフラグが“0”
LT	(S = 1 & O = 0) ¦ (S = 0 & O = 1)	符号付きで小さい

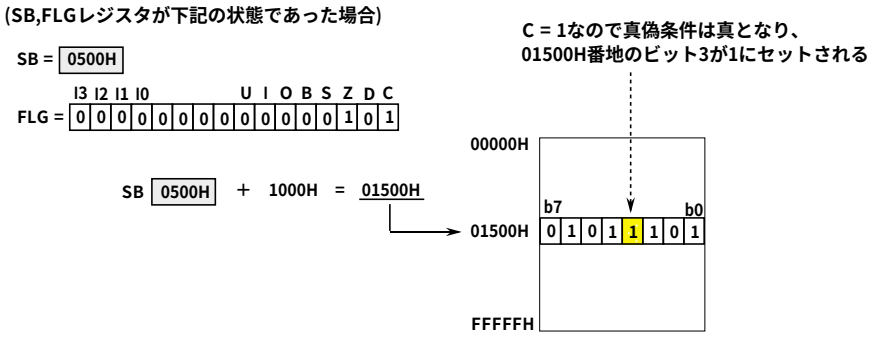


図 2.6.13　条件ビット転送命令の動作例

2.6.6 分岐命令

分岐命令は 10 命令用意されています。この項では特長のある分岐命令を説明します。

無条件分岐命令

無条件で、label へ分岐する命令です。

分岐距離指定子は通常省略します。省略するとアセンブル時にアセンブラが分岐距離を判別し最適化します。無条件分岐命令の動作例を図 2.6.14 に示します。

表 2.6.14 無条件分岐命令

ニーモニック	記述形式		説明
JMP	JMP .S .B .W .A	label	labelへ分岐する

分岐範囲: .S +2～+9のPC相対アドレッシングによる分岐(オペランド:0バイト)
.B -127～+128のPC相対アドレッシングによる分岐(オペランド:1バイト)
.W -32767～+32768のPC相対アドレッシングによる分岐(オペランド:2バイト)
.A 20ビット絶対アドレッシングによる分岐(オペランド:3バイト)

```
JMP   LABEL1
```

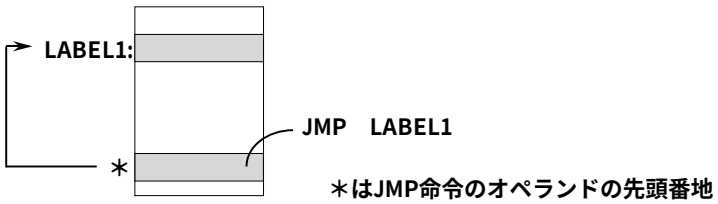


図 2.6.14 無条件分岐命令の動作例

間接分岐命令

src が示す番地に間接分岐する命令です。

分岐距離指定子に .W を指定した場合、JMPI 命令の先頭番地と src を符号付き加算した番地に分岐します。src がメモリのとき、必要なメモリ容量は 2 バイトです。分岐距離指定子に .A を指定した場合は、src に分岐します。src がメモリのとき、必要なメモリ容量は 3 バイトです。この命令の分岐距離指定子は必ず指定してください。間接分岐命令の動作例を図 2.6.15 に示します。

表 2.6.15　間接分岐命令

ニーモニック	記述形式	説明
JMPI	JMPI .W/.A src	srcが示す番地に間接分岐する

分岐範囲 : .W　-32768～+32767のPC相対アドレッシングによる分岐
.A　20ビット絶対アドレッシングによる分岐

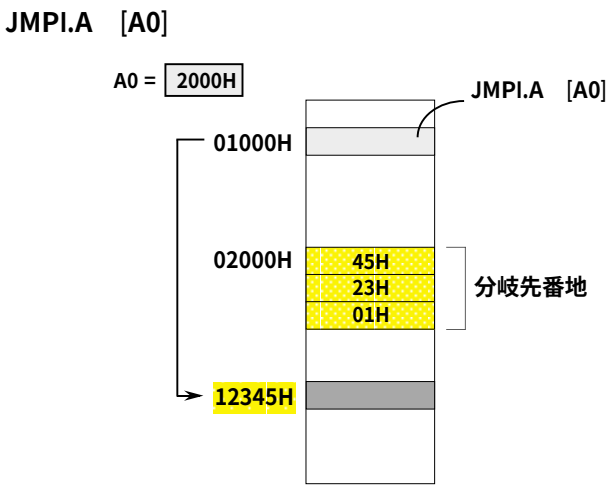


図 2.6.15　間接分岐命令の動作例

スペシャルページ分岐命令

スペシャルページベクタテーブルの各テーブルに設定している番地にF0000Hを加算した番地へ分岐する命令です。分岐範囲はF0000HからFFFFFH番地です。スペシャルページ分岐命令は分岐先番地をメモリに格納しているにもかかわらず、高速分岐を実行します。

分岐先はスペシャルページ番号またはlabelで指定します。スペシャルページ番号の前には“＃”を、labelの前には“￥”を付けてください。なお、labelで指定した場合はアセンブラがスペシャルページ番号を算出します。スペシャルページ分岐命令の動作例を図2.6.16に示します。

表2.6.16　スペシャルページ分岐命令

ニーモニック	記述形式
JMPS	JMPS　#スペシャルページ番号 JMPS　\ラベル 18 ≦ スペシャルページ番号 ≦ 255

JMPS　#251

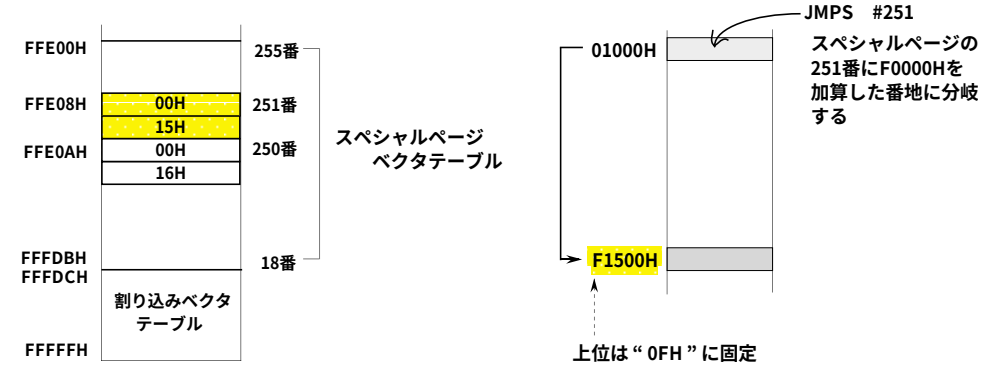


図2.6.16　スペシャルページ分岐命令の動作例

条件分岐命令

条件分岐命令はフラグの状態を下記の条件で判定し、真ならば分岐し、偽ならば次の命令を実行します。条件分岐命令の動作例を図 2.6.17 に示します。

表 2.6.17　条件分岐命令

ニーモニック	記述形式	説明
JCnd	JCnd　label	条件が真ならばlabelへ分岐し、 条件が偽ならば次命令を実行する。

Cnd	真偽判定条件(14種類)	
GEU/C	C = 1	等しいまたは大きい/キャリーフラグが “ 1 ”
GTU	C = 1 & Z = 0	符号なしで大きい
EQ/Z	Z = 1	等しい/ゼロフラグが “ 1 ”
N	S = 1	負
LE	(Z = 1) ¦ (S = 1 & O = 0) ¦ (S = 0 & O = 1)	等しいまたは符号付きで小さい
O	O = 1	オーバフローフラグが “ 1 ”
GE	(S = 1 & O = 1) ¦ (S = 0 & O = 0)	等しいまたは符号付きで大きい
LTU/NC	C = 0	小さい/キャリーフラグが “ 0 ”
LEU	C = 0 ¦ Z = 1	等しいまたは小さい
NE/NZ	Z = 0	等しくない/ゼロフラフが “ 0 ”
PZ	S = 0	正またはゼロ
GT	(S = 1 & O = 1 & Z = 0) ¦ (S = 0 & O = 0 & Z = 0)	符号付きで大きい
NO	O = 0	オーバフローフラグが “ 0 ”
LT	(S = 1 & O = 0) ¦ (S = 0 & O = 1)	符号付きで小さい

分岐範囲:-127 ～ +128(PC 相対)GEU/C,GTU,EQ/Z,N,LTU/NC,LEU,NE/NZ,PZ の場合
　　-126 ～ +129(PC 相対)LE,O,GE,GT,NO,LT の場合

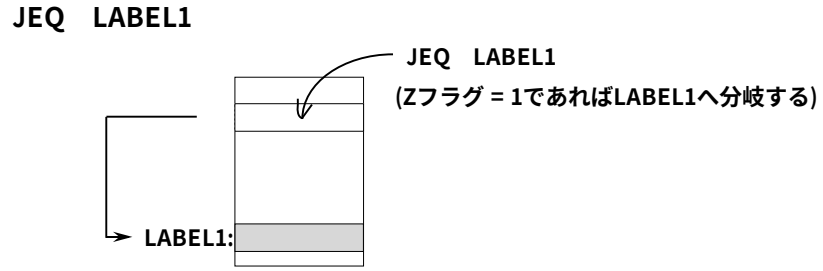


図 2.6.17　条件分岐命令の動作例

2.6.7　高級言語サポート命令

高級言語サポート命令はスタックフレームを構築/解放する命令です。この命令は1命令で、高級言語に対応した複雑な処理を実行します。

スタックフレームの構築

ENTER 命令はスタックフレームを構築する命令です。“ #IMM ” で自動変数領域のバイト数を設定します。スタックフレーム構築命令の動作例を図 2.6.18 に示します。

表 2.6.18　スタックフレーム構築命令

ニーモニック	記述形式	説明
ENTER	ENTER　#IMM	スタックフレームを構築する

(注) #IMMは、IMM8(符号なし8ビット即値)だけで 自動変数領域のサイズ(バイト数)を示します。

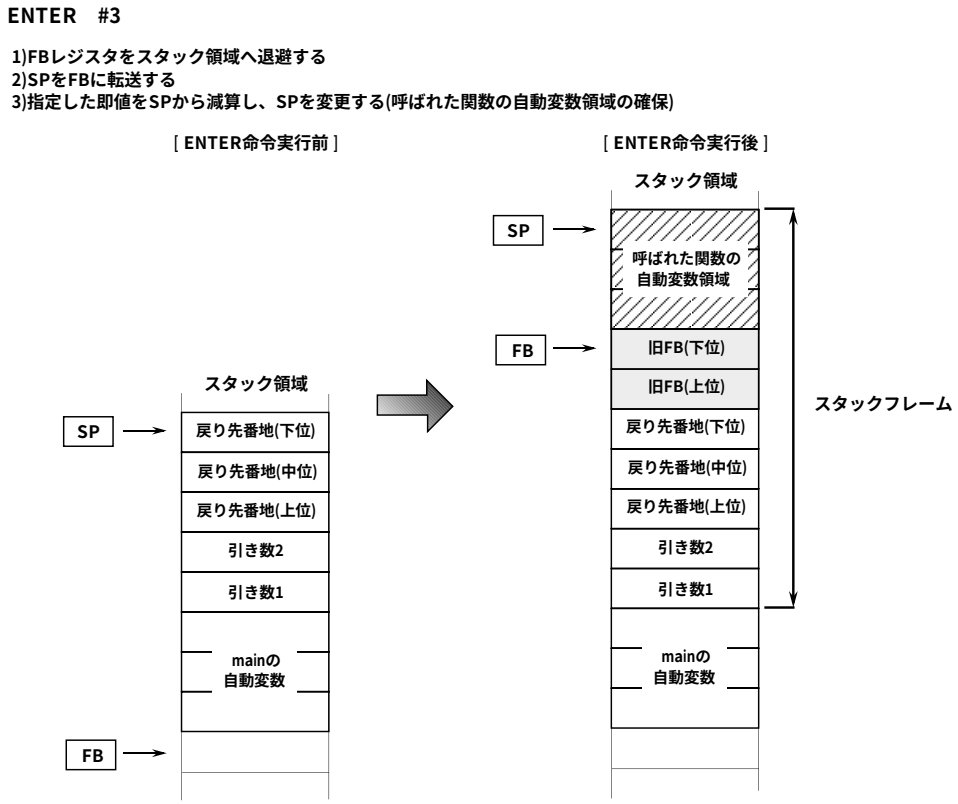


図 2.6.18　スタックフレーム構築命令の動作例

スタックフレームの解放

EXITD命令はスタックフレームの解放とサブルーチンからの復帰を同時に行います。スタックフレーム解放命令の動作例を図 2.6.19 に示します。

表 2.6.19　スタックフレーム解除命令

ニーモニック	記述形式	説明
EXITD	EXITD	スタックフレームを解放する

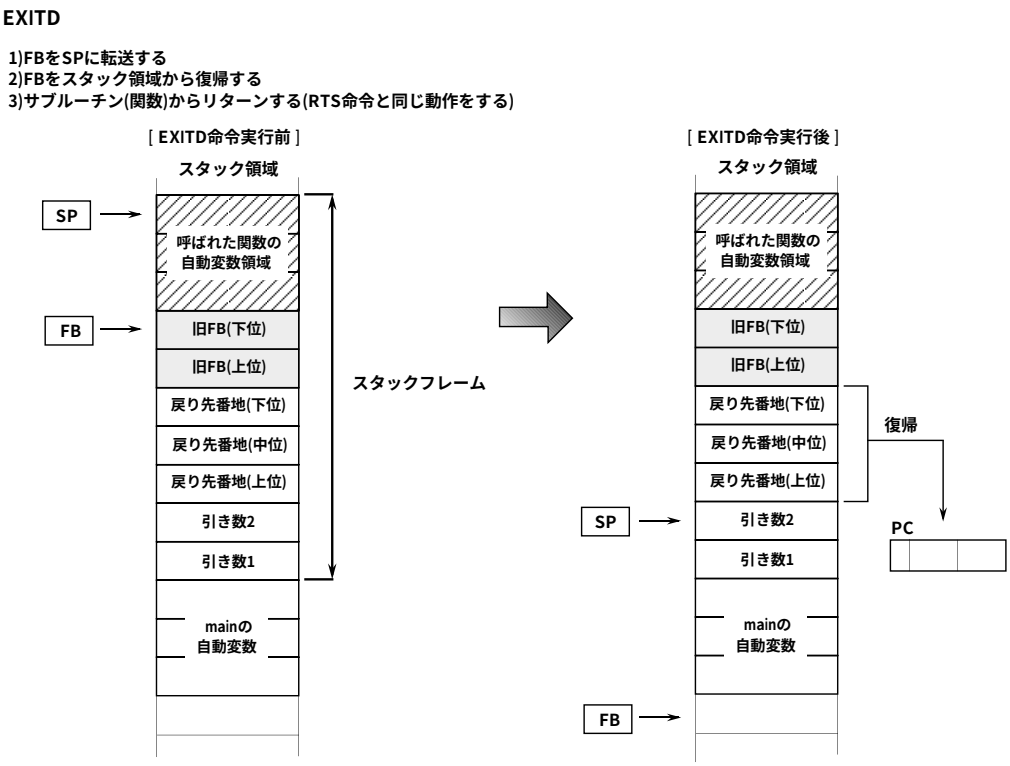


図 2.6.19　スタックフレーム解放命令の動作例

2.6.8　OS サポート命令

OSサポート命令はタスクのコンテキストの退避と復帰を行う命令です。　この命令はタスクの切り替えに必要なコンテキストの切り替えを 1 命令で実行します。

OS サポート命令

STCTX 命令はタスクのコンテキストを退避する命令です。LDCTX 命令はタスクのコンテキストを復帰する命令です。タスクのコンテキストテーブルを、図 2.6.20 に示します。コンテキストテーブルのレジスタ情報にはスタック領域にレジスタの値を転送するかしないかを設定します。SP補正値には転送するレジスタのバイト数を設定します。OS サポート命令は 2 つの情報を使ってタスクのコンテキストをスタック領域に退避、復帰します。

表 2.6.20　OS サポート命令

ニーモニック	記述形式	説明
STCTX	STCTX　abs16,abs20	タスクのコンテキストを退避する
LDCTX	LDCTX　abs16,abs20	タスクのコンテキストを復帰する

(注1) abs16 : タスク番号(8ビット)を格納したメモリ番地
(注2) abs20 : コンテキストテーブルの先頭番地

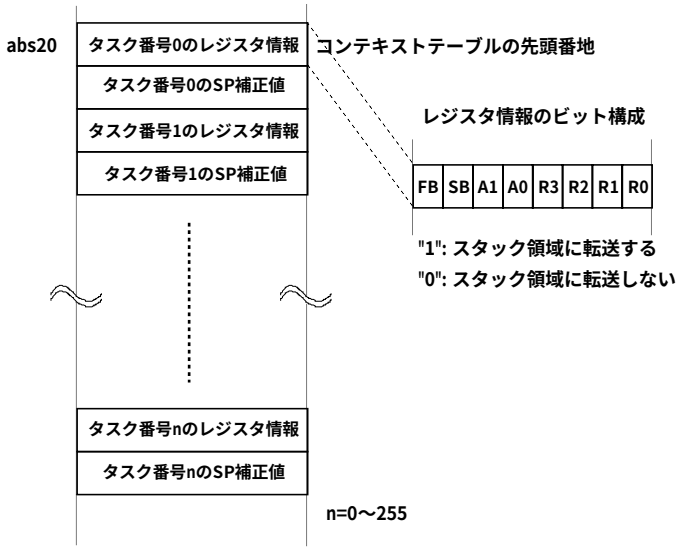
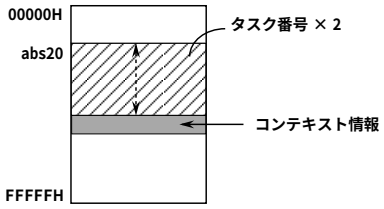


図 2.6.20　コンテキストテーブル

コンテキスト退避の動作（STCTX 命令）

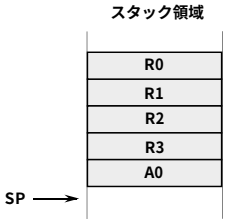
動作 1

abs16(タスク番号)を 2 倍し、
abs20(コンテキストテーブルの先頭番地)を
加算する。 **(タスク番号) × 2 + abs20**
加算結果で示されたメモリ内容を
レジスタ情報(8 ビットデータ)として読み出す。



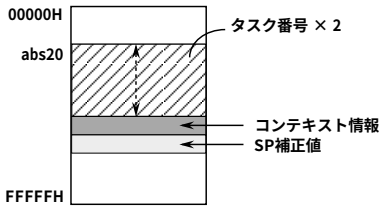
動作 2

レジスタ情報により指定されたレジスタを
スタック領域へ退避する。



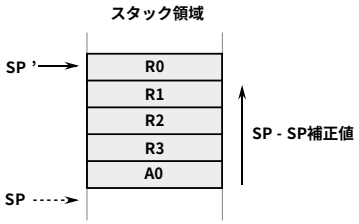
動作 3

レジスタ情報の次の番地(+1 した番地)の内容を
SP 補正値(8 ビットデータ)として読み出す。



動作 4

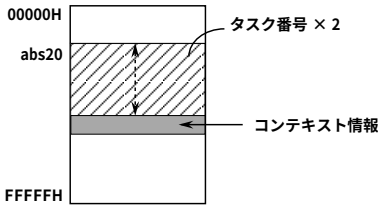
SP から SP 補正値を減算し、SP を変更する。



コンテキスト復帰の動作（LDCTX 命令）

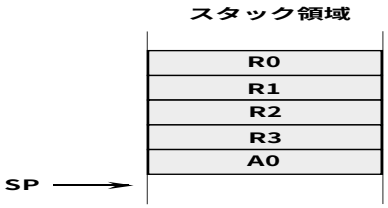
動作 1

abs16(タスク番号)を 2 倍し、
abs20(コンテキストテーブルのベースアドレス)を
加算する。 **(タスク番号) × 2 + abs20**
加算結果で示されたメモリ内容を
レジスタ情報(8 ビットデータ)として読み出す。



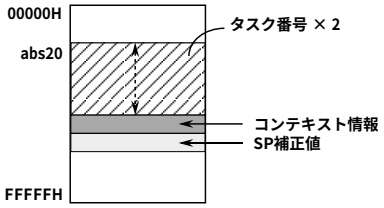
動作 2

レジスタ情報により指定されたレジスタを
スタック領域から復帰する。
(この時点では SP レジスタの値は変化しない)



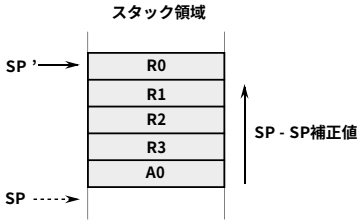
動作 3

レジスタ情報の次の番地(+1 した番地)の内容を
SP 補正値(8 ビットデータ)として読み出す。



動作 4

SP に SP 補正値を加算し、SP を変更する。



2.7　割り込みの概要

割り込み要因の種類と割り込み要求を受け付けてから割り込みルーチンを実行するまでの内部処理（割り込みシーケンス）について説明します。各割り込みの使い方、設定方法については第 4 章をご参照ください。

2.7.1　割り込み要因と制御

この項では M16C/60 グループの割り込み要因について説明します。

M16C/60 グループの割り込み要因

M16C/60 グループの割り込み要因を図 2.7.1 に示します。

ハードウェア割り込みにはリセット、$\overline{\text{NMI}}$ など 6 種類の特殊割り込みと、タイマ、外部端子など内蔵する周辺機能に依存する周辺 I/O 割り込み(注)があります。特殊割り込みはノンマスカブル割り込みです。周辺 I/O 割り込みはマスカブル割り込みです。マスカブル割り込みの許可および禁止は、割り込み許可フラグ(I フラグ)、割り込み優先レベル選択ビット、およびプロセッサ割り込み優先レベル(IPL)によって行います。

ソフトウェア割り込みはソフトウェア割り込み命令の実行によって割り込み要求を発生します。ソフトウェア割り込みには INT 命令割り込み、BRK 命令割り込み、オーバフロー割り込み、未定義命令割り込みの 4 種類があります。

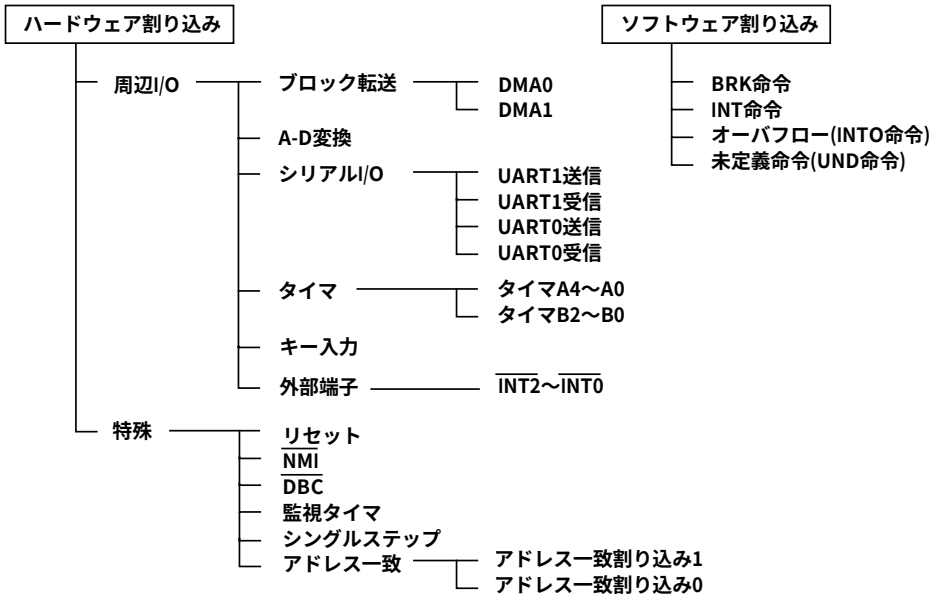


図 2.7.1　M16C/60 グループの割り込み要因

(注)周辺機能は使用する品種によって異なります。周辺割り込みについての詳細はデータシートおよびユーザーズマニュアルを参照してください。

2.7.2　割り込みシーケンス

この項では、割り込みシーケンスについて説明します。

割り込みシーケンス

命令実行中に割り込み要求が発生すると、その命令の実行終了後に優先順位が判定され、次のサイクルから割り込みシーケンスに移ります(図2.7.2)。ただし、ストリング命令(SMOVB,SMOVF,SSTR)と積和演算命令(RMPA)の実行中に割り込み要求が発生すると、命令の動作を一時中断し割り込みシーケンスに移ります(図2.7.3)。

割り込みシーケンスでは、まず割り込み以前のフラグレジスタとプログラムカウンタの内容をスタック領域に退避し、割り込みに関連するレジスタの値(注)を設定します。割り込みシーケンスが終了すると割り込み処理に移ります。なお、割り込みシーケンス実行中はリセット以外の割り込みは受け付けられません。

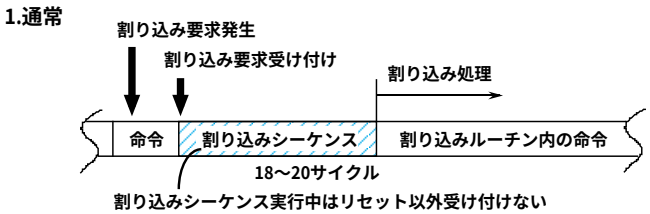


図 2.7.2　割り込みシーケンス 1

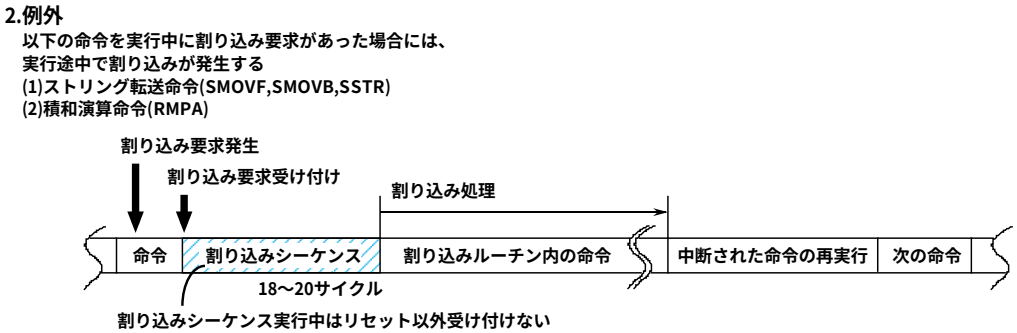


図 2.7.3　割り込みシーケンス 2

(注)フラグレジスタ、プロセッサ割り込み優先レベルなどを示します。

第 3 章

アセンブラの機能

3.1 AS30 システムの概要

AS30 システムは、M16C/60 シリーズ、M16C/20 シリーズシングルチップマイクロコンピュータ制御プログラムの開発を、アセンブリ言語レベルで支援するソフトウェアシステムです。AS30システムにはアセンブラの他に、リンケージエディタ、ロードモジュールコンバータが付属しています。

この節では AS30 の概要を説明します。

機能

●リロケータブルアセンブル機能
●最適化コード生成機能
●マクロ機能
●高級言語のソースレベルデバッグ機能
●各種ファイル生成機能
● IEEE-695 フォーマット (注1) のファイル生成機能

構成

AS30 システムは下記のプログラムで構成しています。

●アセンブラドライバ(as30)
マクロプロセッサ、アセンブラプロセッサを起動する実行ファイルです。アセンブラドライバは、複数のアセンブリソースファイルを処理することができます。

●マクロプロセッサ(mac30)
アセンブリソース中のマクロ指示命令およびアセンブラプロセッサのための前処理を行い、中間ファイルを生成します。中間ファイルはアセンブラプロセッサの処理終了後に消去されます。

●アセンブラプロセッサ(asp30)
マクロプロセッサが生成した中間ファイルをリロケータブルモジュールファイルに変換します。

●リンケージエディタ(ln30)
アセンブラプロセッサの生成したリロケータブルモジュールファイルをリンクし、アブソリュートモジュールファイルを生成します。

●ロードモジュールコンバータ(lmc30) (注2)
リンケージエディタの生成したアブソリュートモジュールファイルをROM化可能な機械語ファイルに変換します。

●ライブラリアン(lb30)
リロケータブルモジュールファイルを読み込み、ライブラリファイルを生成、管理します。

●クロスリファレンサ(xrf30)
ユーザーの作成したアセンブリソースファイル中の各種シンボルおよびラベルの定義情報を格納したクロスリファレンスファイルを生成します。

●アブソリュートリスタ(abs30)
アブソリュートモジュールファイルのアドレス情報を基に、プリントアウト可能なアブソリュートリストファイルを生成します。

(注 1)IEEE(Institute of Electrical and Electronics Engineers:アメリカ電気電子技術者協会)
(注 2)ロードモジュールコンバータはM16C/60 シリーズ、M16C/20 シリーズの ROM に書き込み可能なフォーマッ　トへ変換するプログラムです。

AS30 システムの処理概要

AS30 システムのアセンブル処理の概要を図 3.1.1 に示します。

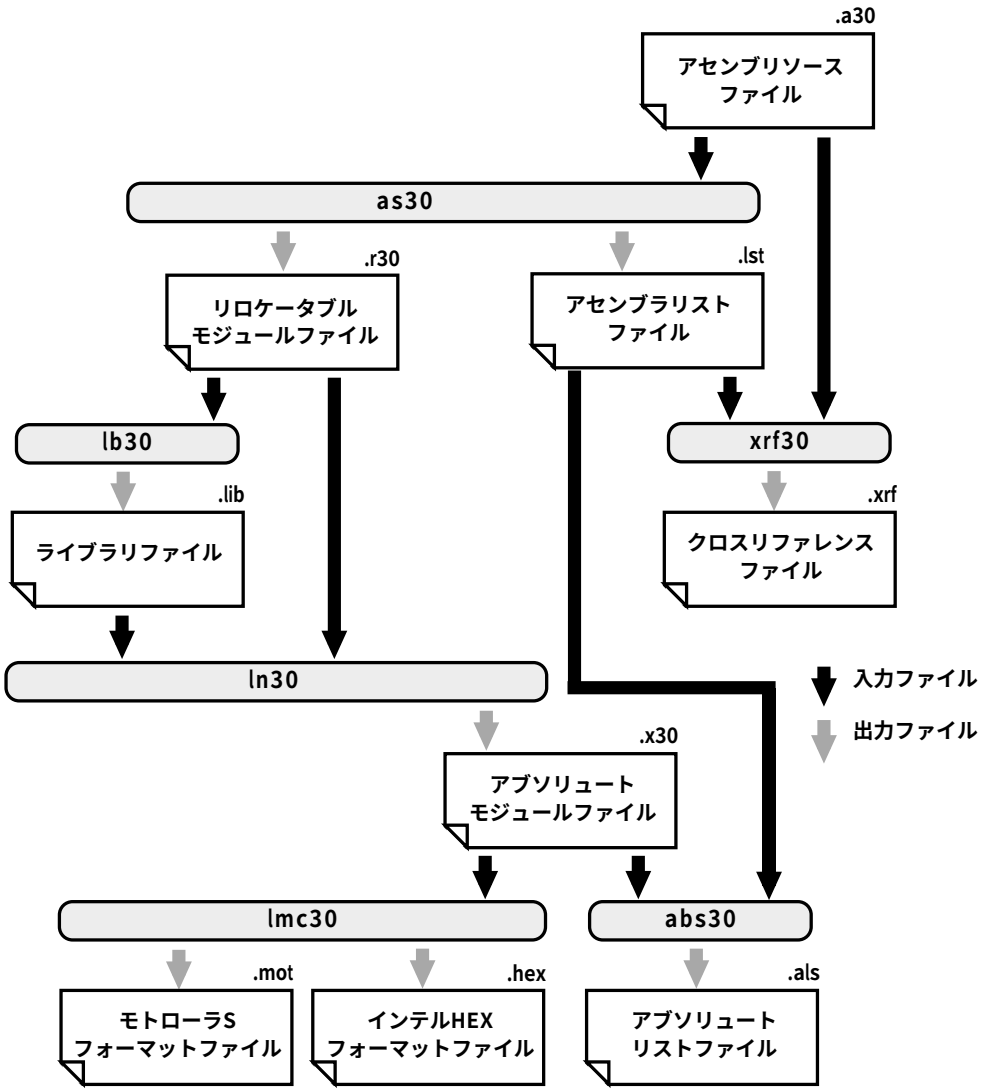


図 3.1.1　AS30 処理概要

AS30 システムの入出力ファイル

AS30システムで扱うファイルを入力ファイルと、出力ファイルに分けて次の表に示します。ファイル名は任意に付けることができます。ただし、ファイル名の拡張子部分については、拡張子を省略すると、AS30システムがデフォルトのファイル拡張子を付加します。この拡張子を表の( )内に記述します。

表 3.1.1 入出力ファイル一覧

プログラム名	入力ファイル名(拡張子)	出力ファイル名(拡張子)
アセンブラ as30	ソースファイル (.as30) インクルードファイル (.inc)	リロケータブルモジュールファイル (.r30) アセンブラリストファイル (.lst) アセンブラエラータグファイル (.atg)
リンケージエディタ ln30	リロケータブルモジュールファイル (.r30) ライブラリファイル (.lib)	アブソリュートモジュールファイル (.x30) マップファイル (.map) リンクエラータグファイル (.ltg)
ロードモジュールコンバータ lmc30	アブソリュートモジュールファイル (.x30)	モトローラSフォーマットファイル (.mot) 拡張インテルHEXフォーマットファイル (.hex)
ライブラリアン lb30	リロケータブルモジュールファイル (.r30) ライブラリファイル (.lib)	ライブラリファイル (.lib) リロケータブルモジュールファイル (.r30) ライブラリストファイル (.lls)
クロスリファレンサ xrf30	アセンブルソースファイル (.a30) アセンブラリストファイル (.lst)	クロスリファレンスファイル (.xrf)
アブソリュートリスタ abs30	アブソリュートモジュールファイル (.x30) アセンブラリストファイル (.lst)	アブソリュートリストファイル (.als)

3.2 ソースプログラムの書き方

AS30 システム対応のソースプログラムを記述する上で必要な事項として基本規則、アドレス管理、指示命令について説明します。AS30 システムについての詳細は AS30 ユーザーズマニュアル << 操作編 >> と << プログラミング編 >> をご参照ください。

3.2.1 基本規則

AS30 システム対応のソースプログラムを記述するための基本規則を説明します。

プログラム記述上の注意事項

AS30 システムを使用する場合には、下記の内容に注意して記述してください。

●予約語は、ソースプログラム中の名前に使用しないでください。
●AS30 システムの指示命令からピリオドをとった文字列は、AS30 の処理に影響する文字列もありますので使用しないでください。、名前に使用してもエラーとなりません。
●システムラベル(..で始まる文字列)については AS30 システムの拡張用に用いられる可能性がありますので使用しないでください。ユーザーがソースプログラム内に記述した場合エラーは出力しません。

文字セット

次に示す文字を使って、AS30 システム対応のアセンブリプログラムを記述できます。なお、命令およびオペランドはすべて半角文字で記述してください。コメント以外には多バイト文字(漢字など)は使用できません。

英文字
A B C D E F G H I J K L M N O P Q R
S T U V W X Y Z
小文字
a b c d e f g h i j k l m n o p q r s t u
v w x y z
数字
0 1 2 3 4 5 6 7 8 9
特殊文字
" # % & ' ( ) * + , - . / : ; [\ ] ^ _ ¦ ~
空白文字
(スペース) (タブ)
改行文字
(リターン) (ラインフィード)

予約語

ＡＳ３０システムの予約語を下記に示します。予約語は、大文字と小文字を区別しません。"abs","ABS","Abs","ABs","AbS","abS","aBs","aBS" はすべて予約語 "ABS" となります。

ニーモニック

ABS	ADC	ADCF	ADD	ADJNZ	AND	BAND
BCLR	BMC	BMEQ	BMGE	BMGEU	BMGT	BMGTU
BMLE	BMLEU	BMLT	BMLTU	BMN	BMNC	BMNE
BMNO	BMNZ	BMO	BMPZ	BMZ	BNAND	BNOR
BNOT	BNTST	BNXOR	BOR	BRK	BSET	BTST
BTSTC	BTSTS	BXOR	CMP	DADC	DADD	DEC
DIV	DIVU	DIVX	DSBB	DSUB	ENTER	EXITD
EXTS	FCLR	FSET	INC	INT	INTO	JC
JEQ	JGE	JGEU	JGT	JGTU	JLE	JLEU
JLT	JLTU	JMP	JMPI	JMPS	JN	JNC
JNE	JNO	JNZ	JO	JPZ	JSR	JSRI
JSRS	JZ	LDC	LDCTX	LDE	LDINTB	LDIPL
MOV	MOVA	MOVHH	MOVHL	MOVLH	MOVLL	MUL
MULU	NEG	NOP	NOT	OR	POP	POPC
POPM	PUSH	PUSHA	PUSHC	PUSHM	REIT	RMPA
ROLC	RORC	ROT	RTS	SBB	SBJNZ	SHA
SHL	SMOVB	SMOVF	SSTR	STC	STCTX	STE
STNZ	STZ	STZX	SUB	TST	UND	WAIT
XCHG	XOR					

レジスタ／フラグ

A0	A1	A1A0	B	C	D	FB
FLG	I	INTBL	INTBH	IPL	ISP	O
PC	R0	R0H	R0L	R1	R1H	R1L
R2	R2R0	R3	R3R1	S	SB	SP
U	USP	Z				

その他

SIZEOF	TOPOF					
IF	ELIF	ELSE	ENDIF	FOR	NEXT	WHILE
ENDW	SWITCH	CASE	DEFAULT	ENDS	REPEAT	UNTIL
BREAK	CONTINUE	FOREVER				

システムラベル(".." で始まる全ての名前)

名前の記述

名前はソースプログラムの中で、任意に定義し使用できます。

名前は次の 4 種類に分けられます。種類によって記述できる範囲が異なります。なお、名前には予約語を使用することはできません。(注)

- **ラベル**
- **シンボル**
- **ビットシンボル**
- **ロケーションシンボル**

名前の記述規則

①使用文字は英数字と "_"(アンダースコア)で、名前の長さは 255 文字までです。
②大文字と小文字を区別します。
③先頭文字に数字は使用できません。

(注)万一使用した場合のプログラムの動作については保証できません。

名前の種類

名前の定義方法を表 3.2.1 に示します。

表 3.2.1　ユーザーが定義する名前の種類

ラベル

機能
　特定のメモリアドレスを示します。

定義方法
　名前の最後に必ず":"(コロン)を付けます。
　定義方法は二つの方法があります。
　　1.指示命令で、領域を確保する。
　　例)

```
flag:    .BLKB   1
work:    .BLKB   1
```

　　2.ソース行の先頭に名前を記述する。
　　例)

```
name1:
_name:
sum_name:
```

参照方法
　命令のオペランドに名前を記述します。
　　例)

```
JMP     sym_name
```

シンボル

機能
　定数値を示します。

定義方法
　数値定義の指示命令を使用します。
　　例)

```
value1  .EQU    1
value2  .EQU    2
```

参照方法
　命令のオペランドにシンボルを記述します。
　　例)

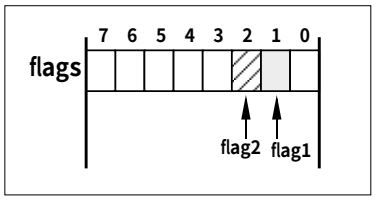
```
MOV.W  R0,value2+1
value3  .EQU    value2+1
```

ビットシンボル

機能
　特定のメモリのビットアドレスを示します。

定義方法
　ビットシンボル定義の指示命令を使用します。
　　例)

```
flag1   .BTEQU 1,flags
flag2   .BTEQU 2,flags
flag3   .BTEQU 20,flags
```



参照方法
　1ビット操作命令のオペランドに記述できます。
　　例)

```
BCLR    flag1
BCLR    flag2
BCLR    flag3
```

ロケーションシンボル

機能
　現在の行を示します。

定義方法
　必要ありません。

参照方法
　ドルマーク($)をオペランドに記述することで、記述した行のアドレスを示します。
　　例)

```
JMP     $+5
```

オペランドの記述

ニーモニックおよび指示命令には、その命令の制御の対象を示す、オペランドを記述します。オペランドは記述方法によって5種類に分けられます。命令によってオペランドを持たない場合もあります。オペランドの有無や使用するオペランドの種類については、各命令の記述方法を参照してください。

●数値

数値は10進数、16進数、2進数、および8進数で記述できます。それぞれの記述方法と記述例を表3.2.2に示します。

表 3.2.2　オペランドの記述

種類	記述例	内容
2進数	10010001B 10010001b	末尾に'B'または'b'を記述します。
8進数	60702o 60702O	末尾に'O'または'o'を記述します。
10進数	9423	末尾には何も記述しません。
16進数	0A5FH 5FH 0a5fh 5fh	0～9、a～fまたはA～Fのいずれかで記述し、末尾に'H'、または'h'を記述します。ただし、アルファベットではじまる数値の場合は、先頭に'0'を付けます。
浮動小数点数	3.4E35 3.4E-35 -.5e20 5e20	指数部には`E`または`e`の後に符号付きで指数を記述します。3.4×10^{35}は3.4E35と記述します。
名前	loop	ラベル名、シンボル名をそのまま記述します。
式	256/2 label/3	数値、名前および演算子を組み合わせて記述します。
文字列	"string" 'string'	シングルまたはダブルクォーテーションで囲って記述してください。

●浮動小数点数

浮動小数点数で表される次の範囲の値を記述できます。浮動小数点数の記述方法と記述例を表3.2.2に示します。浮動小数点数は、指示命令".DOUBLE"、".FLOAT"のオペランドだけに使用できます。それぞれの指示命令で記述できる範囲を表 3.2.3 に示します。

表 3.2.3　浮動小数点数の記述範囲

指示命令	記述範囲
FLOAT(32 ビット長)	**$1.17549435 \times 10^{-38} \sim 3.40282347 \times 10^{38}$**
DOUBLE(64 ビット長)	**$2.2250738585072014 \times 10^{-308} \sim 1.7976931348623157 \times 10^{308}$**

●名前

ラベル名、シンボル名が記述できます。名前の記述方法と記述例を表 3.2.2 に示します。

●式

数値、名前および演算子を組み合わせた式を記述できます。演算子は複数組み合わせて記述できます。シンボル値として式を記述する場合は、式の値がアセンブ ル時に確 定するように式を記述してください。式の演算結果の値は -2147483648 ～ 2147483648 となります。浮動小数点数は式に記述できません。なお、式の記述方法と記述例を表 3.2.2 に示します。

●文字列

一部の指示命令のオペランドに文字列が記述できます。文字列では、7 ビット長 ASCII コードを使用します。シングルまたはダブルクォーテーションで囲って記述してください。文字列の記述方法と記述例を表 3.2.2 に示します。

演算子

AS30 のソースプログラムに記述できる演算子を表 3.2.4 に示します。

表 3.2.4　演算子一覧

単項演算子	
＋	正の値
－	負の値
~	論理否定値
SIZEOF	セクションのサイズ(バイト数)
TOPOF	セクションの開始アドレス

二項演算子	
＋	加算
－	減算
＊	乗算
／	除算
%	割った余り
>>	右へビットシフト
<<	左へビットシフト
&	論理積
¦	論理和
^	排他的論理和

条件演算子	
>	左辺値が右辺値より大きい
<	右辺値が左辺値より大きい
>=	左辺値が右辺値より大きいか等しい
<=	右辺値が左辺値より大きいか等しい
==	左辺値と右辺値が等しい
!=	左辺値と右辺値が等しくない

演算優先順位変更演算子	
()	()で囲った演算を最優先でおこないます。一つの式に複数の()が記述されている場合は、左が優先になります。()はネスティングができます。

(注 1)演算子 "SIZEOF" 及び "TOPOF" はオペランドとの間に、スペースまたはタブを記述してください。

(注 2)条件演算子は、指示命令 ".IF" 及び ".ELIF" のオペランドだけに記述できます。

演算優先順位

演算は演算子の優先順位の高いものから演算します。優先順位を表3.2.5に示します。優先順位が等しい場合には、式の左から順に演算します。演算の順番は(　)で囲うことで変更できます。

表 3.2.5　演算優先順位

	優先順位	演算子の種類	内容
高 ↑	1	演算順位変更演算子	(,)
	2	単項演算子	＋ , －, ~ , SIZEOF , TOPOF
	3	二項演算子1	＊,／ , %
	4	二項演算子2	＋, －
	5	二項演算子3	>> , <<
	6	二項演算子4	&
	7	二項演算子5	¦ , ^
↓ 低	8	条件演算子	> , < , >= , <= , == , !=

行の記述

AS30はソースプログラムを行単位で処理します。改行文字で区切られ、改行文字の直後の文字から、次の改行文字までを1行とします。1行に記述可能な最大文字数は255文字です。行は記述されている内容によって5種類に分けられます。それぞれの記述方法を表3.2.6に示します。

- ●指示命令行
- ●アセンブリソース行
- ●ラベル定義行
- ●コメント行
- ●空行

表 3.2.6　行の種類

指示命令行

機能
as30の指示命令を記述した行です。

記述方法
指示命令は、一行に一つのみ記述できます。
指示命令行には、コメントを記述できます。
注意事項
指示命令とニーモニックを同じ一行に記述することはできません。

例)

```
        .SECTION    program,DATA
        .ORG        00H
sym     .EQU        0
work:   .BLKB       1
        .ALIGN
        .PAGE       "newpage"
        .ALIGN
```

アセンブリソース行

機能
ニーモニックを記述した行です。

記述方法
アセンブリソース行には、ラベル名（先頭）、コメントを記述できます。
注意事項
１行には、２つ以上のニーモニックは記述できません。
指示命令とニーモニックを同じ一行に記述することはできません。

例)

```
        MOV.W       #0,R0
        RTS
main:   MOV.W       #0,A0
        RTS
```

ラベル定義行

機能
ラベル名だけを記述した行です。

記述方法
ラベル名には、必ず続けてコロン(:)を記述してください。

例)

```
start:
label:  .BLKB   1
main:   nop
loop:
```

コメント行

機能
コメントだけを記述した行です。

記述方法
コメントの先頭には、必ずセミコロン(;)を記述してください。

例)

```
        ;コメント行
        MOV.W       #0,A0
```

空行

機能
見かけ上何も記述していない行です。

記述方法
スペース、タブまたは、改行コードだけを記述してください。

3.2.2　アドレス管理

この項では AS30 システムのアドレス制御方法について説明します。

AS30 システムはアドレス制御を行なう際に RAM サイズと ROM サイズは考慮しません。したがって実際に使用するシステムにおけるアドレス範囲を考えて、ソースプログラムの記述やリンク処理を行なってください。

アドレスの管理方法

AS30システムは、セクション単位でアドレスを管理します。セクションの区切りは、次のように決められます。なお、セクションのネスティングは定義できません。

セクションの区切り

①指示命令 ".SECTION" を記述した行から、次の指示命令 ".SECTION" を記述した前の行までの間

②指示命令 ".SECTION" を記述した行から、指示命令 ".END" を記述した前の行までの間

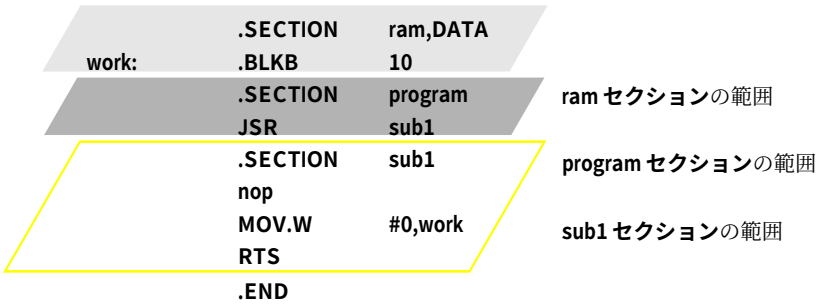


図 3.2.1　AS30 システムにおけるセクションの範囲

セクションのタイプ

アドレス制御の単位となるセクションには、タイプを設定できます。タイプによって、セクション内に記述できる命令が異なります。

表 3.2.7　セクションのタイプ

タイプ	内容と記述例
ＣＯＤＥ (プログラム領域)	・プログラムを記述する領域です。 ・領域を確保する指示命令を除く全ての命令が記述できます。 ・CODEタイプのセクションは、アブソリュートモジュールにおいて、ROM領域に配置されるように指定してください。 例) .SECTION　program,CODE
ＤＡＴＡ (データ領域)	・内容が変更可能なメモリを配置する領域です。 ・領域を確保する指示命令が記述できます。 ・DATAタイプのセクションは、アブソリュートモジュールにおいて、RAM領域に配置されるように指定してください。 例) .SECTION　mem,DATA
ＲＯＭＤＡＴＡ (固定データ領域)	・プログラム以外の固定データを記述する領域です。 ・データを設定する指示命令が記述できます。 ・ROMDATAタイプのセクションは、アブソリュートモジュールにおいて、ROM領域に配置されるように指定してください。 例) .SECTION　const,ROMDATA

セクションの属性

アドレス制御の単位となるセクションは、アセンブル時に属性が付けられます。

表 3.2.8　セクションの属性

属性	内容と記述例
相対	・アセンブル時にセクション内のアドレスがリロケータブル値となります。 ・相対属性セクション内で定義されたラベルの値はリロケータブルです。
絶対	・アセンブル時にセクション内のアドレスがアブソリュート値となります。 ・絶対属性セクション内で定義されたラベルの値はアブソリュートです。 ・セクションを絶対属性にするためには、指示命令".SECTION " を記述した次の行で、指示命令".ORG"でアドレスを指定してください。 例) .SECTION　program,DATA .ORG　1000H

セクションの整列

相対属性のセクションでは、リンク時に決定するセクションのスタートアドレスが必ず偶数番地になるように調整することができます。調整を行ないたいときは、指示命令 ".SECTION" のオペランドに "ALIGN" を指定するか、指示命令 ".ALIGN" を指示命令 ".SECTION" の次の行に記述してください。

例)

```
.SECTION        program,CODE,ALIGN
または

.SECTION        program,CODE
.ALIGN
```

AS30 システムのアドレス管理

AS30 システムが、複数のファイルにわたって記述されたアセンブリソースプログラムを、1 つの実行形式のファイルに変換する方法を次に示します。

as30 のアドレス管理

①絶対属性となるセクションは、指定されているアドレスから順に絶対アドレスを決定します。
②相対属性となるセクションは、セクション毎に 0 から順にアドレスを決定します。相対属性のセクションの開始アドレスはすべて 0 です。

ln30 のアドレス管理

①全てのファイルの同一名のセクションを、指定されたファイルの順に並べます。
②まとめたセクションの先頭から順に絶対アドレスを決定します。
③セクションの開始アドレスは、指定がなければ 0 から順に決定します。
④異なるセクションとセクションは、指定がなければ連続したアドレスに配置します。

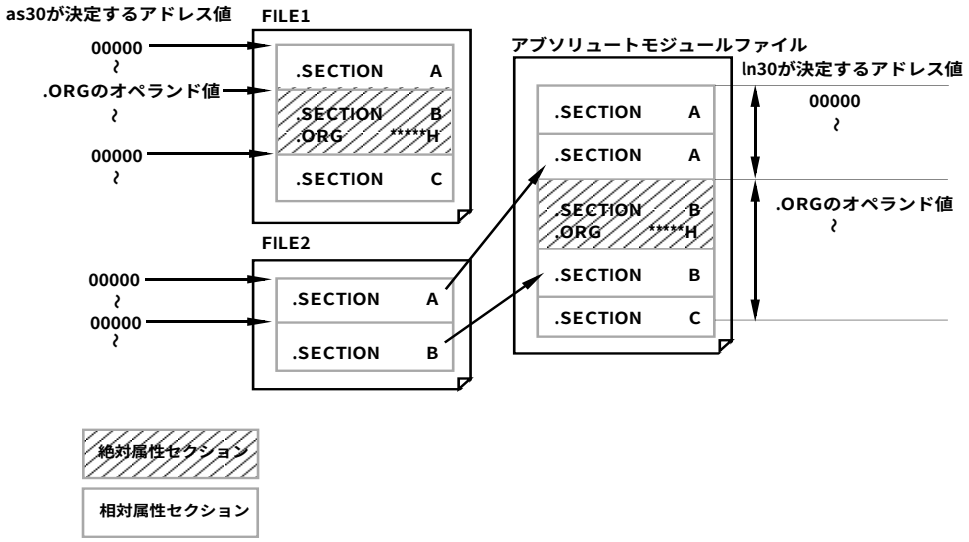


図 3.2.2　アドレス管理の例

インクルードファイルの読み込み

AS30 システムは、ソースプログラムの任意の行で、インクルードファイルを読み込むことができます。プログラムの可読性の向上などに利用できます。

インクルードファイルの読み込み

指示命令".INCLUDE" のオペランドに読み込みたいファイル名を記述します。この行　の位置にインクルードファイルの内容が全て読み込まれます。

例)

```
.INCLUDE        initial.inc
```

ソースファイル(sample.a30)

```
        .SECTION    memory,DATA
work:   .BLKB   10
flags:  .BLKW   1
        .SECTION    init
        .INCLUDE    initial.inc
        .SECTION    program,CODE
main:
            :
        .END
```

インクルードファイル(initial.inc)

```
loop:
        MOV.W       #10,A0
        MOV.B       #0,work[A0]
        INC.W       A0
        JNZ         loop
        MOV.W       #0,flags
```

展開イメージ

アセンブル実行後

```
                            .SECTION    memory,DATA
000000          work:       .BLKB   10
00000A          flags:      .BLKW   1
                            .SECTION    init
000000                      .INCLUDE    initial
000000          loop:   MOV.W       #10,A0
000002                  MOV.B       #0,work[A0]
000006                  INC.W       A0
000007                  JNZ         loop
000009                  MOV.W       #0,flags

000000                      .SECTION    program,CODE
                main:
                            :
                            .END
```

as30が出力したアドレス

図 3.2.3　インクルードファイルの読み込み

グローバルとローカルのアドレス管理

AS30 システムにおけるラベル、シンボル及びビットシンボルの値の管理について説明します。
AS30 システムでは、ラベル、シンボル及びビットシンボルの値をグローバルとローカル、リロケータブルとアブソリュートに分けて扱います。以下にそれぞれの定義を示します。

●グローバル

指示命令".GLB"で指定したラベル及びシンボルは、それぞれグローバルラベル及びグローバルシンボルとなります。
指示命令 ".BTGLB" で指定したビットシンボルはグローバルビットシンボルとなります。
ファイル内に定義がある名前をグローバル指定したものは、外部のファイルからの参照が可能になります。
ファイル内に定義のない名前をグローバル指定したものは、外部のファイルで定義されている名前を参照する外部参照ラベル、シンボル、ビットシンボルとなります。

●ローカル

指示命令 ".GLB" または ".BTGLB" で指定のない名前は、すべてローカルとなります。
ローカルな名前は、定義した同一ファイル内でだけ参照できます。
ローカルな名前は、別のファイルで同一のラベル名を使用できます。

●リロケータブル

相対セクション内のローカルラベル、シンボル、ビットシンボルの値は、リロケータブルになります。
外部参照となるグローバルラベル、シンボル、ビットシンボルの値は、リロケータブルとなります。

●アブソリュート

絶対属性セクション内に定義のあるローカルラベル、シンボル、ビットシンボルの値は、アブソリュートとなります。

アブソリュートとなるラベル、シンボル、ビットシンボルは、as30 が値を決定します。その他のラベル、シンボル、ビットシンボルの値は、すべてリンク時に ln30 が決定します。
ラベルの関係を図 3.2.4 に示します。

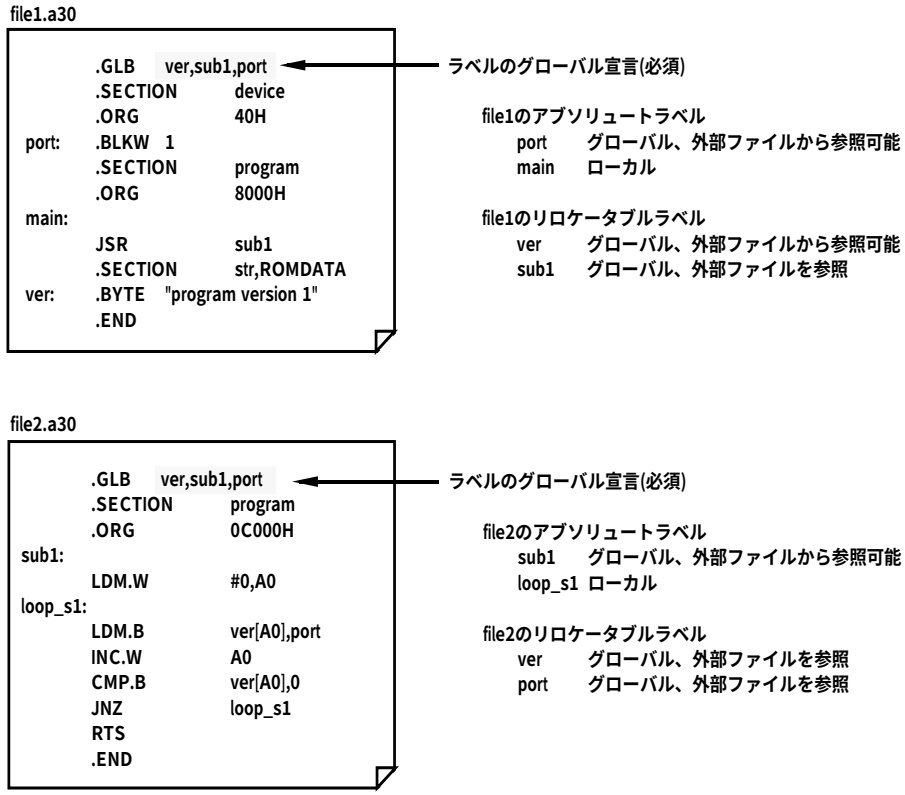

file1.a30
.GLB ver,sub1,port
.SECTION device
.ORG 40H
port: .BLKW 1
.SECTION program
.ORG 8000H
main:
JSR sub1
.SECTION str,ROMDATA
ver: .BYTE "program version 1"
.END
ラベルのグローバル宣言(必須)
file1のアブソリュートラベル
port グローバル、外部ファイルから参照可能
main ローカル
file1のリロケータブルラベル
ver グローバル、外部ファイルから参照可能
sub1 グローバル、外部ファイルを参照
file2.a30
.GLB ver,sub1,port
.SECTION program
.ORG 0C000H
sub1:
LDM.W #0,A0
loop_s1:
LDM.B ver[A0],port
INC.W A0
CMP.B ver[A0],0
JNZ loop_s1
RTS
.END
ラベルのグローバル宣言(必須)
file2のアブソリュートラベル
sub1 グローバル、外部ファイルから参照可能
loop_s1 ローカル
file2のリロケータブルラベル
ver グローバル、外部ファイルを参照
port グローバル、外部ファイルを参照

図 3.2.4　ラベルの関係

3.2.3 指示命令

ソースプログラムには、M16C/60 シリーズ、M16C/20 シリーズの機械語命令の他に AS30 システムの指示命令を使用できます。指示命令には下記の種類があります。指示命令の使い方について、種類別に説明します。

●アドレス制御命令

アセンブル実行時にアドレス決定の指示ができます。

●アセンブル制御指示命令

AS30 の実行について指示できます。

●リンク制御指示命令

アドレス再配置制御のための情報を定義できます。

●リスト制御指示命令

AS30 が生成するリストファイルのフォーマットを制御できます。

●分岐最適化制御指示命令

分岐命令の最適選択を AS30 に指示できます。

●条件アセンブル制御指示命令

アセンブル実行時に設定した条件によって、コード生成するブロックを選択できます。

●拡張機能指示命令

上記以外の制御を行なう指示命令です。

● M16C ファミリ用ツールソフトウェアが出力する指示命令

これらの指示命令及びオペランドについては、全て M16C ファミリ用ツールソフトウェアが出力します。ユーザーはソースプログラムに記述できません。

アドレス制御

命令	機能	使い方･記述例
.ORG	アドレスを宣言します。	セクション指示命令".SECTION"の直後に記述してください。セクション指示命令の直後にない場合、そのセクションは相対属性セクションとなります。相対属性のセクション内には記述できません。 .ORG 0F0000H .ORG offset .ORG 0F0000H + offset
.BLKB	1バイト単位でRAM領域を確保します。	DATAセクション内に確保する領域数を記述してください。ラベル名を定義する場合は必ずコロン(:)を付けてください。 例) .BLKB 1 .BLKW number .BLKA number+1 label: .BLKL 1 label: .BLKF number label: .BLKD number+1
.BLKW	2バイト単位でRAM領域を確保します。	
.BLKA	3バイト単位でRAM領域を確保します。	
.BLKL	4バイト単位でRAM領域を確保します。	
.BLKF	4バイト単位でRAM領域を浮動小数点数用に確保します。	
.BLKD	8バイト単位でRAM領域を確保します。	
.BYTE	1バイト長データでROM領域に格納します。	複数のオペランドを記述するときは、カンマ(,)で区切ってください。ラベル名を定義する場合は必ずコロン(:)を付けてください。 .FLOAT と.DOUBLEについてはオペランドに浮動小数点数を記述してください。 例) .SECTION value,ROMDATA .BYTE 1 .BYTE 1,2,3,4,5 .WORD "da","ta" .ADDR symbol .LWORD symbol+1 .FLOAT 5E2 constant .DOUBLE 5e2
.WORD	2バイト長データでROM領域に格納します。	
.ADDR	3バイト長データでROM領域に格納します。	
.LWORD	4バイト長データでROM領域に格納します。	
.FLOAT	4バイト長データでROM領域に浮動小数点数を格納します。	
.DOUBLE	8バイト長データでROM領域に浮動小数点数を格納します。	
.ALIGN	奇数アドレスを偶数アドレスに補正します。	セクション定義の際にアドレス補正を指示している相対属性セクションか絶対属性セクション内に記述できます。 例) .SECTION program,CODE .ORG 0F000H MOV.W #0,R0 .ALIGN .END

アセンブル制御

命令	機能	使い方･記述例
.EQU	シンボルを定義します。	前方参照となるシンボル名は記述できません。オペランドにはシンボル、式を記述できます。シンボル、ビットシンボルはグローバル指定ができます。 例) symbol .EQU 1 symbol1 .EQU symbol+symbol bit0 .BTEQU 0,0 bit1 .BTEQU 1,symbol1
.BTEQU	ビットシンボルを定義します。	
.END	アセンブルソースの終了を宣言します。	一つのアセンブリソースファイルに必ず一つ以上記述してください。as30は本指示命令以降の行については、エラーの検出はしません。 例) .END
.SB	SBレジスタ値を仮定します。	各アドレッシングモードを選択する前に必ず各レジスタを設定してください。 レジスタ値の設定は実際のレジスタには設定しませんので、本命令の直前または直後にレジスタ値を設定する命令を記述してください。 例) .SB 80H LDC #80H,SB .FB 0C0H LCD #80H,FB .SBSYM sym1,sym2 .FBSYM sym3,sym4
.SBSYM	SB相対アドレッシングを選択します。	
.SBBIT	ビット命令のSB相対アドレッシングを選択します。	
.FB	FBレジスタ値を仮定します。	
.FBSYM	FB相対アドレッシングを選択します。	
.INCLUDE	ファイルを指定位置に読み込みます。	オペランドのファイル名には必ず拡張子を記述してください。オペランドには、指示命令"..FILE"や"@"を含む文字列が記述できます。 例) .INCLUDE initial.a30 .INCLUDE ..FILE@.inc

リンク制御

命令	機能	使い方･記述例
.SECTION	セクション名を定義します。	セクションタイプ及びALIGNを同時に指定する場合はカンマで区切ってください。セクションタイプは、'CODE','ROMDATA','DATA'のいずれかを記述できます。セクションタイプを省略した場合は、"CODE"として処理します。 例) .SECTION program,CODE NOP .SECTION ram,DATA .BLKB 10 .SECTION dname,ROMDATA .BYTE "abcd" .END
.GLB	グローバルラベルを指定します。	オペランドに複数のシンボル名を記述する場合はカンマ(,)で区切ってください。 例) .GLB name1,name2,mane3 .BTGLB flag4 .SECTION program MOV.W #0,name1 BCLR flag4
.BTGLB	グローバルビットシンボルを指定します。	
.VER	指定した文字列をマップファイルにバージョン情報として出力します。	オペランドは、1行の範囲内で記述してください。1つのアセンブリソースファイルに1度しか記述できません。 例) .VER 'strings' .VER "strings"

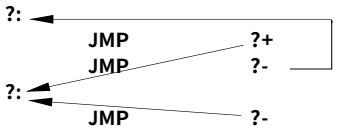
リスト制御

命令	機能	使い方・記述例
.LIST	リストファイルへの行データの出力を制御します。	行出力を停止する場合は'OFF'、開始する場合は'ON'をオペランドに'記述してください。指定しない場合はすべての行をリストファイルに出力します。 例) .LIST OFF MOV.B #0,R0L MOV.B #0,R0L MOV.B #0,R0L .LIST ON
.PAGE	リストファイルの指定位置で改ページを行ないます。	オペランドは、クォーテーション(')またはダブルクォーテーション(")で囲って記述してください。オペランドは省略できます。 例) .PAGE .PAGE "strings" .PAGE 'strings'
.FORM	リストファイルの1ページの桁及び行数を指定します。	1つのアセンブリソースファイルに複数回記述できます。行数及び桁数にはシンボルを記述できます。前方参照となるシンボルは記述できません。出力しない場合は66行、140桁で出力します。 例) .FORM 20,80 .FORM 60 .FORM ,100 .FORM line,culmn

分岐命令最適化制御

命令	機能	使い方・記述例
.OPTJ	分岐命令とサブルーチン呼び出しの最適化を制御します。	オペランドには、分岐命令の最適制御、最適化対象外の無条件分岐命令選択、最適化対象外のサブルーチン呼び出し命令選択の項目が記述できます。各項目の指定順序は任意で、省略もできます。省略した場合は分岐距離は初期値または以前に指定した内容から変化しません。 例) 下記に示す組み合わせのオペランドが記述できます。 .OPTJ OFF .OPTJ ON .OPTJ ON,JMPW .OPTJ ON,JMPW,JSRW .OPTJ ON,JMPA .OPTJ ON,JMPA,JSRW .OPTJ ON,JMPA,JSRA .OPTJ ON,JMRW .OPTJ ON,JMRA

拡張機能指示命令

命令	機能	使い方･記述例
.ASSERT	ファイルまたは標準エラー出力に指定した文字列を出力します。	ダブルクォーテーションで囲った文字列をファイルに出力する場合は、">"または">>"に続けてファイル名を指定してください。 >は新規にファイルを生成して、そのファイルにメッセージを出力します。以前に同一名のファイルが或る場合は、そのファイルに上書きされます。 >>はファイルの内容に追加して、メッセージを出力します。指定したファイルが存在しない場合は、新しくファイルを生成します。 ファイル名に指示命令"..FILE"を記述できます。 例) .ASSERT "string" > sample.dat .ASSERT "string" >> sample.dat .ASSERT "string" > ..FILE
?	テンポラリラベルの指定及び参照を行ないます。	テンポラリラベルとして定義したい行に"?:"を記述してください。直前に定義したテンポラリラベルを参照したい場合は、命令のオペランドに"?-"、直後に定義したテンポラリラベルを参照したい場合は、命令のオペランドに"?+"を記述してください。 例) 
..FILE	ソースファイル名情報を示します。	指示命令".ASSERT"及び指示命令".INCLUDE"のオペランドに記述できます。コマンドオプション"-F"を指定すると"..FILE"はコマンド行で指定したソースファイル名に固定されます。オプションを指定しない場合は"..FILE"が記述されているファイル名になります。 例) .ASSERT "sample" > ..FILE .INCLUDE ..FILE@.inc .ASSERT "sample" > ..FILE@.mes
@	@の前後の文字列を連結します。	1行に複数回記述できます。 連結した文字列を名前とする場合は、本命令の前後にスペース及びタブを記述しないでください。 例) .ASSERT "sample" > ..FILE@.dat 次のようなマクロ定義も可能です。 mov_nibble .MACRO p1,src,p2,dest MOV@p1@p2 src,dest .ENDM

条件アセンブル指示命令

命令	機能	使い方･記述例
.IF	条件アセンブルの開始を示します。	オペランドには必ず条件式を記述してください。 例) .IF TYPE==0 .BYTE "Proto Type Mode" .ELIF TYPE>0 .BYTE "Mass Production Mode" .ELSE .BYTE "Debug Mode" .ENDIF 条件式の記述規則) 演算結果のオーバフロー及びアンダーフローは判断しません。シンボルは前方参照(本指示命令行より後に定義されているシンボルを参照)はできません。前方参照のシンボルや、未定義のシンボルを記述した場合は、値を0として式を判断します。 条件式の記述例) sym<1 sym < 1 sym+2 < data1 sym+2 < data1+2 'smp1' ==name
.ELIF	条件アセンブルの条件を示します。	必ずオペランドには条件式を記述してください。本指示命令は1つの条件アセンブルブロック内に複数記述できます。 例) 同上
.ELSE	条件偽の際にアセンブルを行なうブロックの開始を示します。	本指示命令は、条件アセンブルブロック内に1つ以下記述できます。オペランドはありません。 例) 同上
.ENDIF	条件アセンブルの終了を示します。	本指示命令は、条件アセンブルブロック内に必ず1つ記述してください。オペランドはありません。 例) 同上

M16C ファミリ用ツールが出力する指示命令

命令	機能	使い方･記述例
"._"で始まる名前	M16Cファミリ用ツールソフトウェアが出力します。	ユーザーはソースプログラムに記述できません、記述した場合の動作については保証しません。 例) ._FILE

3.2.4 マクロ機能

AS30 で使用できるマクロ機能について説明します。AS30 には以下に示すマクロ機能があります。

●マクロ機能

マクロ指示命令".MACRO"～".ENDM"で定義し、定義したマクロを呼び出すことでマクロ機能を使用できます。

●繰返しマクロ機能

マクロ指示命令 ".MREPEAT" ～ ".ENDM" を記述することで、繰返しマクロ機能を使用できます。

マクロ定義とマクロ呼び出しの関係を図 3.25 に示します。

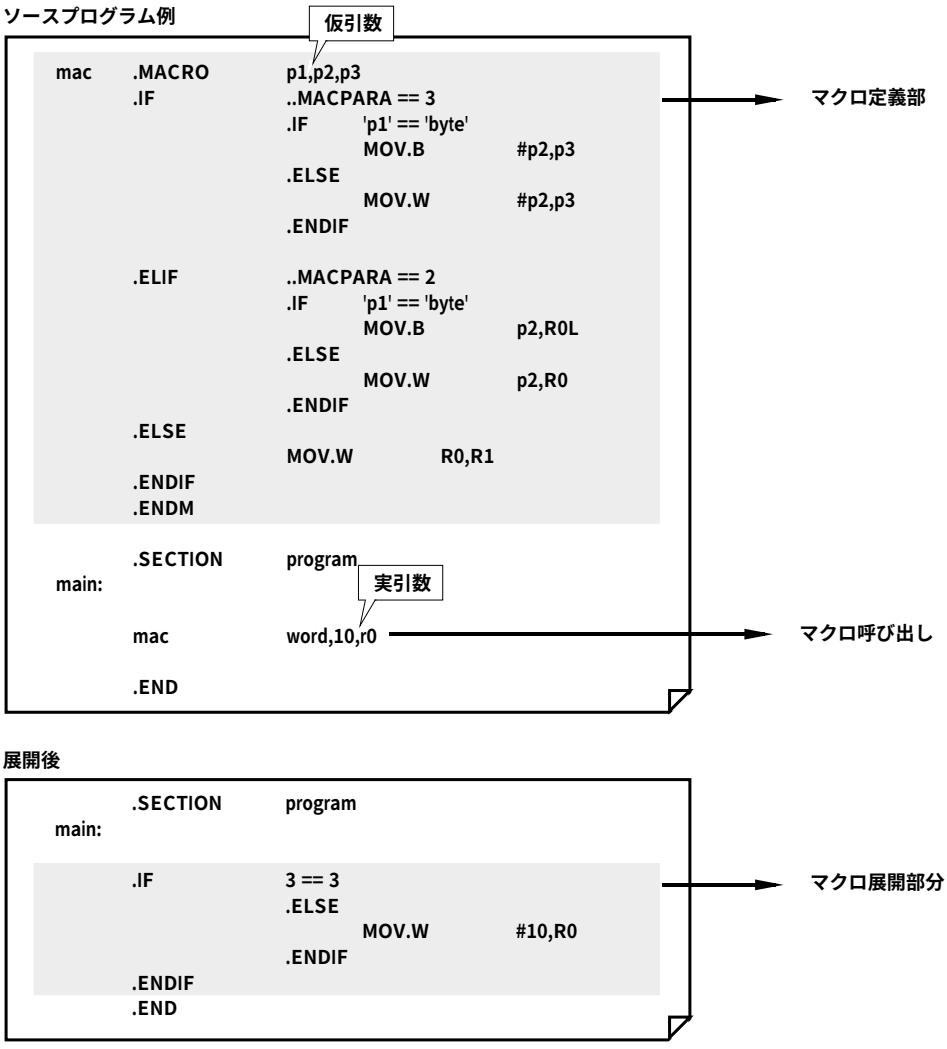


図 3.2.5　マクロ定義と呼び出し例

マクロ定義

マクロ定義はマクロ指示命令".MACRO"を使用して１行以上の命令の集まりを１つのマクロ名に定義します。定義終了は".ENDM"で示し、".MACRO" と ".ENDM" に囲まれた行をマクロボディと呼びます。

マクロボディにはソースプログラムに記述可能なすべての命令を記述できますが、ビットシンボルは記述できません。
マクロのネストはマクロ定義及びマクロ呼び出しを含めて 65535 レベル以下です。
マクロ名及びマクロ引数は大文字、小文字を区別して扱います。

マクロローカル

指示命令".LOCAL"で宣言したマクロローカルラベルは、マクロ定義内のみ使用できます。マクロローカル宣言をしたラベルは、マクロの範囲外で同一のラベルを記述できます。図3.2.6に記述例を示します。この場合、m1がマクロローカルラベルになります。

```
name    .MACRO      source,dest,top
        .LOCLA      m1
m1:
        nop
        jmp         m1

        .ENDM
```

図 3.2.6　マクロ定義と呼び出し例

マクロ呼び出し

マクロ定義されているマクロボディの内容をマクロ指示命令".MACRO"で定義したマクロ名を記述することで、その行に呼び出しを行います。マクロ名の外部参照はできません。複数ファイルから、同一のマクロを呼び出す場合はインクルードファイル内マクロを定義し、そのファイルをインクルードしてください。

繰り返しマクロ機能

マクロ指示命令 ".MREPEAT" ～ ".ENDM" で囲まれたボディを指定した回数分繰り返し、指定した行に展開します。繰返しマクロのマクロ呼び出しはありません。

繰り返しマクロのマクロ定義とマクロ呼び出しの関係を図 3.2.7 に示します。

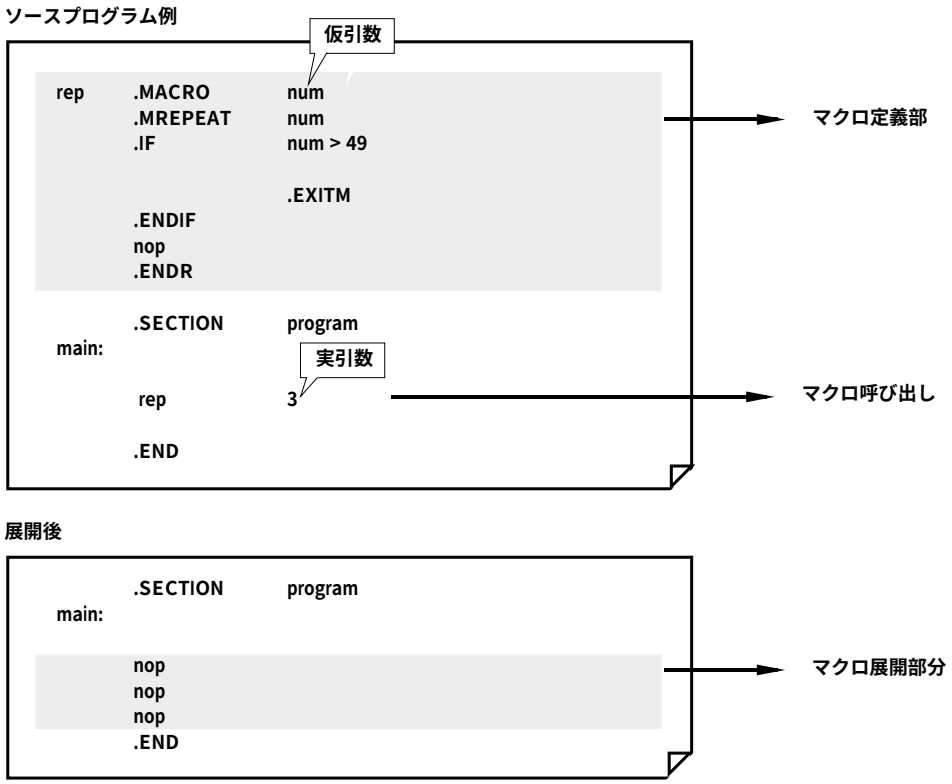


図 3.2.7　マクロ定義と呼び出し例

マクロ指示命令

AS30 のマクロ命令には、以下の種類があります。

●マクロ指示命令
　マクロボディの開始や終了または中止、マクロ内のローカルなどの宣言をします。
●マクロシンボル
　マクロ記述の式の項目として記述します。
●文字列関数
　文字列の情報を表します。

マクロ指示命令

命令	機能	使い方･記述例
.MACRO	マクロ名の定義及びマクロ定義の開始を示します。	オペランドには必ず条件式を記述してください。 仮引数は80個まで記述できます。仮引数をダブルクォーテーションで囲わないでください。 <記述形式> 　マクロ定義 　(マクロ名)　　.MACRO　[(仮引数)[,(仮引数)...]] 　マクロ呼び出し 　(マクロ名)　　[(実引数)[,(実引数)...]] <記述例> 　　図3.2.5を参照してください。
.ENDM	マクロ定義の終了を示します。	".MACRO"に対応させて記述してください。 <記述例> 　　図3.2.5を参照してください。
.LOCAL	オペランドに示されたラベルがマクロローカルラベルであることを宣言します。	マクロボディ内に記述してください。 オペランドはカンマで区切って複数のラベルを記述できます。このとき最大ラベル数は100個です。 <記述例> 　　図3.2.6を参照してください。
.EXITM	マクロボディの展開を強制的に終了します。	マクロ定義のボディ内に記述してください。 <記述例> 　　図3.2.7を参照してください。
.MREPEAT	繰返しマクロ定義の開始を示します。	繰返し回数は最大65535回です。 <記述例> 　　図3.2.7を参照してください。
.ENDR	繰返しマクロ定義の終了を示します。	".MREPEAT"に対応させて記述してください。 <記述例> 　　図3.2.5を参照してください。

マクロシンボル

命令	機能	使い方･記述例
..MACPARA	マクロ呼び出しの際に与えられた実引数の個数を示します。	マクロ定義のボディ内に式の項として記述できます。マクロボディの外に記述した場合、値は０となります。 <記述例> 図3.2.5を参照してください。
..MACREP	繰り返しマクロが展開している回数を示します。	マクロ定義のボディ内に式の項として記述できます。条件アセンブルのオペランドとしても記述できます。繰り返す度に値は１→２→３...と増加します。マクロボディの外に記述した場合、値は０となります。 <記述例> 図3.2.5を参照してください。

文字列関数

命令	機能	使い方･記述例
.LEN	オペランドに記述した文字列長を示します。	オペランドは必ず"{}"で囲い、文字列はクォーテーションで囲ってください。文字列には７ビットアスキーコードの文字が記述できます。 式の項として記述できます。 <記述形式> .LEN {"(文字列)"} .LEN {'(文字列)'} <記述例> 図3.2.8を参照してください。
.INSTR	オペランドで指定した文字列のなかで、検索文字列の開始位置を示します。	オペランドは必ず"{}"で囲い、文字列はクォーテーションで囲ってください。文字列には７ビットアスキーコードの文字が記述できます。 検索開始位置を１にした場合は、文字列の先頭を示します。 <記述形式> .INSTR {"(文字列)","(検索文字列)",(検索開始位置)} .INSTR {'(文字列)','(検索文字列)',(検索開始位置)} <記述例> 図3.2.9を参照してください。
.SUBSTR	オペランドで指定した文字列の位置から、指定した文字数を切り出します。	オペランドは必ず"{}"で囲い、文字列はクォーテーションで囲ってください。文字列には７ビットアスキーコードの文字が記述できます。 切り出し開始位置を１にした場合は、文字列の先頭を示します。 <記述形式> .SUBSTR {"(文字列)",(開始位置),(文字数)} .SUBSTR {'(文字列)',(開始位置),(文字数)} <記述例> 図3.2.10を参照してください。

.LEN 記述例

図 3.2.8 の例では、指定した文字列の文字列長 "Printout_data" は "13"、"Sample" は "6" を表しています。

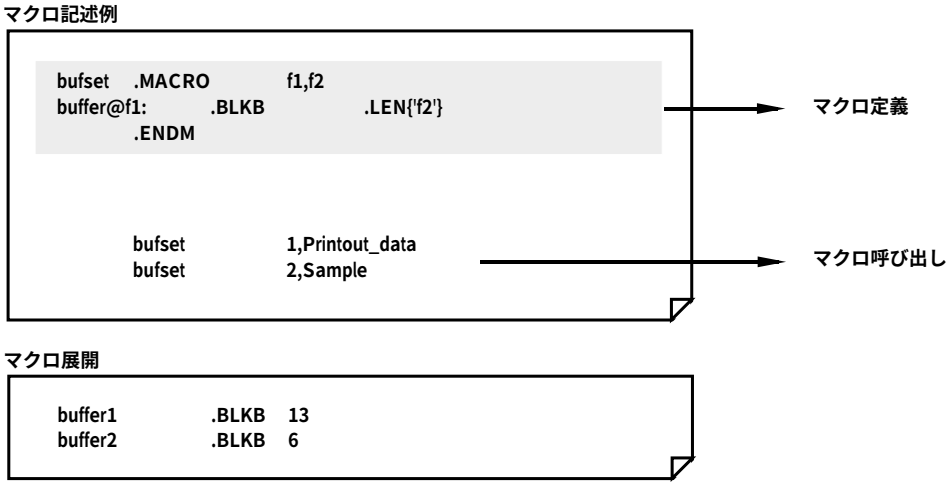


図 3.2.8　.LEN 記述例

.INSTR 記述例

図 3.2.9 の例では、指定した文字列(japanese)の先頭 x (top)からの、"se" 文字列の位置(7)を切り出しています。

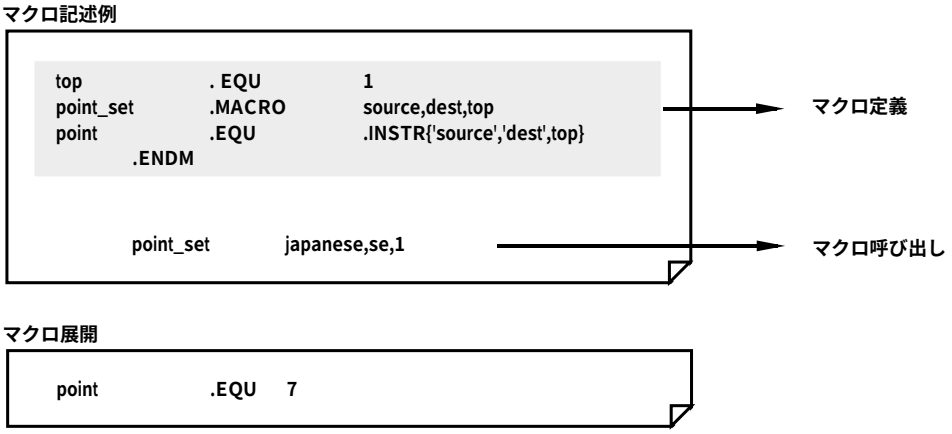


図 3.2.9　.INST 記述例

.SUBSTR 記述例

図 3.2.10 の例では、マクロの実引数として与えられた文字列の長さを、".MREPEAT" のオペランドに与えています。"..MACREP" は ".BYTE" の行を実行するごとに、１→２→３→４ ... と増加します。したがって、マクロ実引数として与えた文字列の先頭文字から順に１文字づつ ".BYTE' のオペランドに与えることになります。

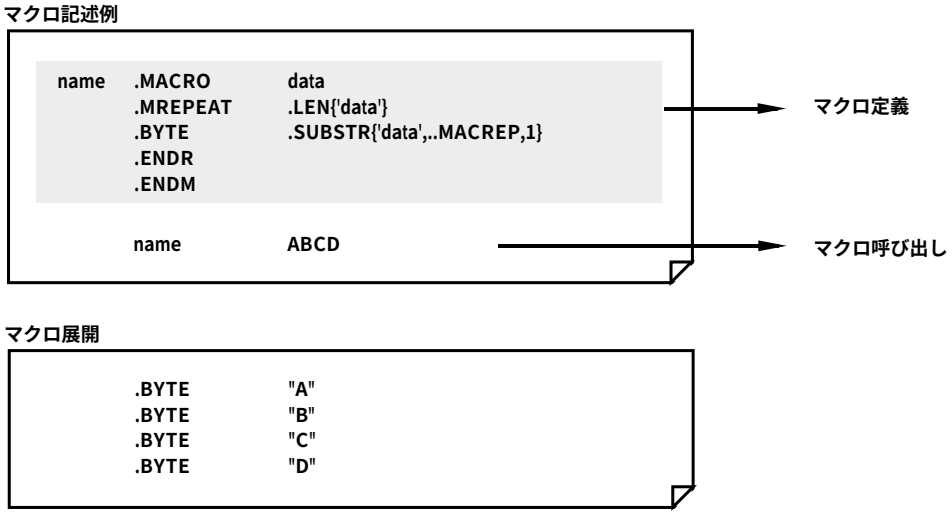


図 3.2.10　.SUBSTR 記述例

3.2.5　構造化記述機能

AS30 のプログラミングでは、構造化命令を使用した構造化記述が可能です。
本書では構造化記述機能についての概要のみ記載します。詳細はAS30ユーザーズマニュアル<<プログラミング編 >> を参照ください。

以下に AS30 の構造化記述機能を示します。

- ***アセンブラは、構造化記述命令に対応するアセンブリ言語のブランチ命令を生成します。***
- ***アセンブラは、生成したブランチ命令の分岐先のラベルを生成します。***
- ***アセンブラは、構造化記述命令から生成したアセンブリ言語を、アセンブラリストファイルに出力します（コマンドオプション指定時）***
- ***構造化記述命令は、構造化記述文とその条件式によって分岐する制御ブロックを選択できます。制御ブロックとは、代入文を除くある構造化記述文から次の構造化記述文までの間を言います。***

構造化記述文の種類

AS30 では、次に示す 9 種類の文が記述できます。

代入文
右辺を左辺に代入します。

IF ELIF ELSE ENDIF 文（以降 IF 文と記す）
IF 文は制御の流を 2 方向に変える命令で、分岐する方向は条件式によって決定されます。

FOR NEXT 文（以降 FOR-NEXT 文と記す）
FOR-NEXT 文は繰り返しを制御する命令で、指定した条件式が真である間、文を繰り返し実行します。

FOR TO STEP NEXT 文（以降 FOR-STEP 文と記す）
FOR-STEP 文は初期値、増分と最終値を指定することで繰り返し回数を制御する命令です。

DO WHILE 文（以降 DO 文と記す）
DO 文は条件式が満たされている（真である）間、文を繰り返し実行します。

SWITCH CASE DEFAULT ENDS 文（以降 SWITCH 文と記す）
SWITCH 文は条件式の値によっていずれかの CASE ブロックに分岐します。

BREAK 文
BREAK 文は該当する FOR 文、DO 文又は SWITCH 文の実行を中止して、その次に実行する文に分岐します。

CONTINUE 文
CONTINUE 文はそれを含む最小の繰り返しの FOR 文、DO 文中の繰り返しを行う文に分岐します。

FOREVER 文
FOREVER 文は該当する FOR 文及び DO 文の条件式を常に真であると仮定して制御ブロックを繰り返し実行します。

第 4 章

プログラミングスタイル

4.1　ハードウェアの定義

この節ではSFR領域の定義とインクルードファイルの作成方法、RAMデータ領域の確保、ROMデータ領域の確保、セクションの定義について説明します。

4.1.1　ＳＦＲ領域の定義

SFR領域の定義部分はインクルードファイルで作成すると便利です。SFR領域の定義方法には以下の２通りがあります。

.EQU による定義

指示命令 ".EQU" を使った SFR 領域の定義例を図 4.1.1 に示します。

```
;-----------------------------------------------------------------
;          M30600 SFR 定義ファイル
;-----------------------------------------------------------------
PM0       .EQU          0004H          ;プロセッサモードレジスタ 0
PM1       .EQU          0005H          ;プロセッサモードレジスタ 1
CM0       .EQU          0006H          ;システムクロック制御レジスタ 0
CM1       .EQU          0007H          ;システムクロック制御レジスタ 1
CSR       .EQU          0008H          ;チップセレクト制御レジスタ
AIER      .EQU          0009H          ;アドレス一致割り込み許可レジスタ
PRCR      .EQU          000AH          ;プロテクトレジスタ
;
WDTS      .EQU          000EH          ;監視タイマスタートレジスタ
WDC       .EQU          000FH          ;監視タイマ制御レジスタ
RMAD0     .EQU          0010H          ;アドレス一致割り込みレジスタ 0
RMAD1     .EQU          0014H          ;アドレス一致割り込みレジスタ 1
;
SAR0      .EQU          0020H          ;DMA0 ソースポインタ
DAR0      .EQU          0024H          ;DMA0 ディスティネーションポインタ
TCR0      .EQU          0028H          ;DMA0 転送カウンタ
DM0CON    .EQU          002CH          ;DMA0 制御レジスタ
SAR1      .EQU          0030H          ;DMA1 ソースポインタ
DAR1      .EQU          0034H          ;DMA1 ディスティネーションポインタ
TCR1      .EQU          0038H          ;DMA1 転送カウンタ
DM1CON    .EQU          003CH          ;DMA1 制御レジスタ
;
```

プロセッサモードレジスタ 0 の配置されているアドレスを定義する
以後各レジスタのアドレスを定義する

２バイト以上のレジスタについてはレジスタの先頭アドレスを定義する

図 4.1.1　".EQU" による SFR 領域定義例

.BLKB による定義

指示命令 ".BLKB" を使った SFR 領域の定義例を図 4.1.2 に示します。

```
;-----------------------------------------------------------------
;           M30600 SFR 定義ファイル
;-----------------------------------------------------------------
  .SECTION      SFR,DATA
  .ORG   00004H
PM0:            .BLKB       1       :プロセッサモードレジスタ 0
PM1:            .BLKB       1       ;プロセッサモードレジスタ 1
CM0:            .BLKB       1       ;システムクロック制御レジスタ 0
CM1:            .BLKB       1       ;システムクロック制御レジスタ 1
CSR:            .BLKB       1       ;チップセレクト制御レジスタ
AIER:           .BLKB       1       ;アドレス一致割り込み許可レジスタ
PRCR:           .BLKB       1       ;プロテクトレジスタ
;
  .ORG   0000EH
WDTS:           .BLKB       1       ;監視タイマスタートレジスタ
WDC:            .BLKB       1       ;監視タイマ制御レジスタ
RMAD0:          .BLKA       1       ;アドレス一致割り込みレジスタ 0
                .BLKB       1       ;
RMAD1:          .BLKA       1       ;アドレス一致割り込みレジスタ 1
                .BLKB       1       ;
;
  .ORG   00020H
SAR0:           .BLKA       1       ;DMA0 ソースポインタ
                .BLKB       1       ;
DAR0:           .BLKA       1       ;DMA0 ディスティネーションポインタ
                .BLKB       1       ;
TCR0:           .BLKW       1       ;DMA0 転送カウンタ
                .BLKB       2       ;
DM0CON:         .BLKB       1       ;DMA0 制御レジスタ
                .BLKB       3       ;
SAR1:           .BLKA       1       ;DMA1 ソースポインタ
                .BLKB       1       ;
DAR1:           .BLKA       1       ;DMA1 ディスティネーションポインタ
                .BLKB       1       ;
TCR1:           .BLKW       1       ;DMA1 転送カウンタ
                .BLKB       2       ;
DM1CON:         .BLKB       1       ;DMA1 制御レジスタ
;
```

セクション名の宣言

プロセッサモードレジスタ 0 の配置されているアドレスに合わせて絶対アドレスを指定

プロセッサモードレジスタ 0 の配置されている領域分確保

ここで絶対アドレスを 0000EH 番地に指定しないと監視タイマスタートレジスタの領域がプロテクトレジスタの次、0000BH に設定されてしまうので注意!!

何も配置されていない部分の領域も確保

図 4.1.2　".BLKB" による SFR 領域定義例

インクルードファイルの作成

ソースプログラムを分割して作成する場合、SFR定義など複数のファイルで使用する部分はインクルードファイルにします。インクルードファイルには通常 ".INC" という拡張子を付けます。

作成上の注意

(1)インクルードファイル中に ".EQU" を使用する場合

".EQU" はシンボルに値を定義する命令です。SFR定義のようにアドレスを定義することもできます。ただし、領域確保命令ではないのでアドレスを定義する場合は、アドレスが重ならないように注意してください。".EQU" を使用したインクルードファイルは複数のファイルで読み込むことができます。

(2)インクルードファイル中に ".ORG" を使用する場合

".ORG"を使用したインクルードファイルを複数のファイルで読み込むとリンクエラーになります。これは ".ORG" によって絶対アドレスが指定されているため、アドレスが 2 重定義となります。

(3)インクルードファイル中に ".BLKB"、".BLKW"、".BLKA" を使用している場合

".BLKB"、".BLKW"、".BLKA" は領域確保命令です。".BLKB"、".BLKW"、".BLKA"を使用したインクルードファイルを複数のファイルで読み込むと領域を別々に確保することになります。インクルードファイルのシンボルをローカルで使用している場合はエラーではありませんが、グローバルで使用している場合は 2 重定義になります。

複数のファイルで共通領域として利用したい場合は、領域確保している部分は共有定義ファイルとし、ソースファイルの 1 つとしてリンクします。そして、シンボルのグローバル指定部分をインクルードファイルにします。

インクルードファイルの読み込み

インクルードファイルを読み込むときは指示命令".INCLUDE"を使います。読み込むファイル名はフルネームで指定してください。

例)

SFR 領域の定義を行ったインクルードファイル "M30600.INC ” を読み込む場合

```
.INCLUDE     M30600.INC
```

4.1.2　RAM データ領域の確保

領域確保は以下の指示命令を使用します。

.BLKB	・・・	1 バイトの領域確保（整数値）
.BLKW	・・・	2 バイトの領域確保（整数値）
.BLKA	・・・	3 バイトの領域確保（整数値）
.BLKL	・・・	4 バイトの領域確保（整数値）
.BLKF	・・・	4 バイトの領域確保（浮動小数点数）
.BLKD	・・・	8 バイトの領域確保（浮動小数点数）

ワーク領域の設定例 4.1　ハードウェアの定義

ワーク領域の設定例を図 4.1.3 に示します。

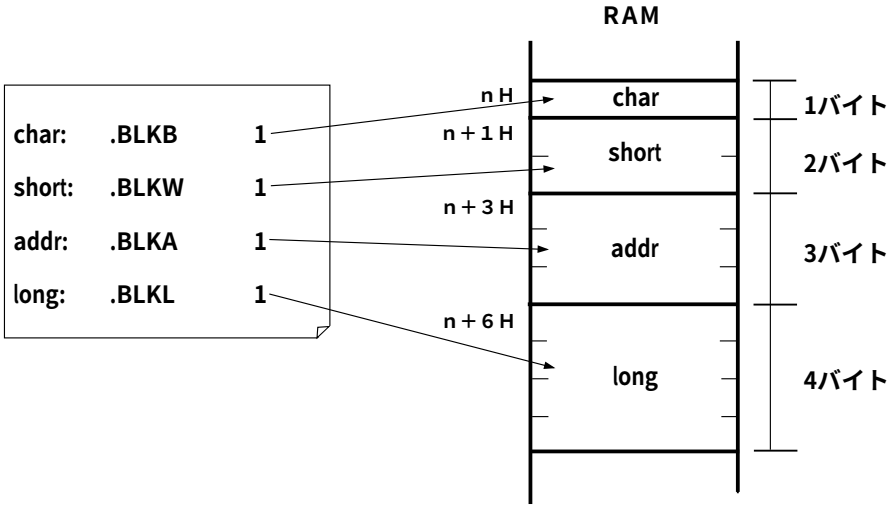


図 4.1.3　ワーク領域の設定例

4.1.3　ROM データ領域の確保

固定データの設定は以下の指示命令を使用します。記述例は「4.1.5　サンプルリスト 1(初期設定 1)」をご参照ください。

.BYTE　・・・　1 バイトデータの設定（整数値）
.WORD　・・・　2 バイトデータの設定（整数値）
.ADDR　・・・　3 バイトデータの設定（整数値）
.LWORD　・・・　4 バイトデータの設定（整数値）
.FLOAT　・・・　4 バイトデータの設定（浮動小数点数）
.DOUBLE　・・・　8 バイトデータの設定（浮動小数点数）

テーブルデータの検索

データデーブルの例を図4.1.4に示します。このテーブルをアドレスレジスタ相対アドレッシングを用いてアクセスする方法を図 4.1.5 に示します。

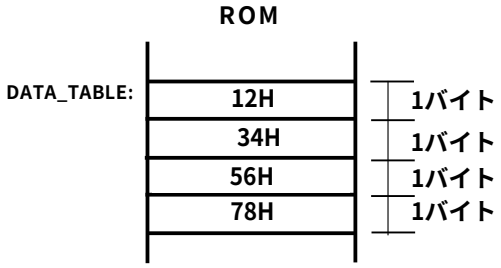


図 4.1.4　データテーブルの設定例

```
        .
        .
        .
        MOV.W     #1,A0
        LDE.B     DATA_TABLE[A0],R0L    ;データテーブルの2バイト目(34H)を
                                        ;R0Lに格納
        .
        .
        .
        .
        .
DATA_TABLE:
        .BYTE 12H,34H,56H,78H           ;1バイトデータの設定
        .
        .
        .
```

図 4.1.5　データテーブルの検索例

4.1.4　セクション定義

指示命令".SECTION"は、この指示命令を記述した行から次の".SECTION"までを割り付けるセクションを宣言します。

セクション定義の記述形式

.SECTION　セクション名［,（セクションタイプ）, ALIGN］
［］内は省略可能

１つのセクション指示命令から次のセクション指示命令､又は".END"を記述した前の行までを１つのセクションとして定義します。セクション名は自由に設定できます。また、各セクションにはセクションタイプ（DATA,CODE,ROMDATA の 3 タイプ）が設定できます。このタイプによってセクション内に記述できる命令が異なりますのご注意ください。詳しくは AS30 ユーザーズマニュアル＜プログラミング編＞をご参照ください。
なお "ALIGN" 指定がある場合、リンカ(ln30)がセクションの始まりを偶数番地に割り当てます。

各セクションの設定例

セクションの設定例を図 4.1.6 に示します。

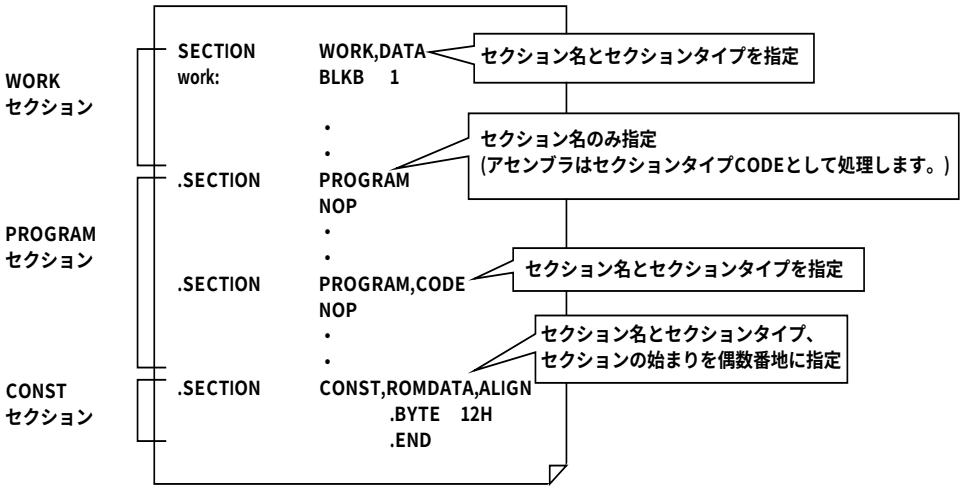


図 4.1.6　セクションの設定例

セクションの属性

各セクションはアセンブル時に属性が割り付けられます。属性は相対と絶対の２つです。

(1)相対属性

・リンク時に各セクションの配置を指定できます。（再配置可能）
・アセンブル時にセクション内のアドレスがリロケータブル値になります。
・相対属性セクション内で定義されたラベルの値はリロケータブルとなります。

(2)絶対属性

・SECTION" 指示命令の直後に ".ORG" によりアドレスを指定することで、そのセクションは絶対属性として扱われます。
・アセンブル時にセクション内のアドレスがアブソリュート値になります。
・絶対属性セクション内で定義されたラベルの値はアブソリュートとなります。

4.1.5　サンプルリスト 1(初期設定 1)

```
;********************** インクルード **********************************************
;
  .INCLUDE              m30600.inc
;
************************ シンボル定義 ***********************************************
;
RAM_TOP             .EQU   00400H       ; RAM の先頭アドレス
RAM_END             .EQU   02BFFH       ; RAM の終了アドレス
ROM_TOP             .EQU   0F0000H      ; ROM の先頭アドレス
FIXED_VECT_TOP      .EQU   0FFFDCH      ;固定ベクタの先頭アドレス
SB_BASE             .EQU   00380H       ; SB 相対のベースアドレス
FB_BASE             .EQU   00480H       ; FB 相対のベースアドレス
;
.************************ ワーク RAM 領域確保 ***************************************
  .SECTION              WORK,DATA
  .ORG          RAM_TOP                 ;
;
WORKRAM_TOP:
char:               .BLKB       1       ;1 バイト領域確保
short:              .BLKW       1       ;2 バイト領域確保
addr:               .BLKA       1       ;3 バイト領域確保
long:               .BLKL       1       ;4 バイト領域確保
WORKRAM_END:
;
;********************** ビットシンボル定義 ****************************************
;
char_b0             .BTEQU      0,char      ; char のビット 0
short_b1            .BTEQU      1,short     ; short のビット 1
addr_b2             .BTEQU      2,addr      ; addr のビット 2
long_b3             .BTEQU      3,long      ; long のビット 3
;
;********************** プログラム領域 ********************************************
;=================== スタートアップ ======================================
;
  .SECTION      PROGRAM,CODE          ;セクション名、セクションタイプの宣言
  .ORG          ROM_TOP               ;開始アドレスの宣言
  .SB           SB_BASE               ; SB レジスタの値をアセンブラに対して宣言
  .FB           FB_BASE               ; FB レジスタの値をアセンブラに対して宣言
;
START:
        LDC     #RAM_END+1,ISP        ;スタックポインタの初期値設定
        LDC     #SB_BASE,SB           ; SB レジスタ初期値設定
        LDC     #FB_BASE,FB           ; FB レジスタ初期値設定
```

インクルードファイルの取込み

ラベルは名前の最後に ":"(コロン)を付ける

ハードウェアの RAM 領域に合わせる

ビットシンボルは ":"(コロン)を付けない

アセンブラに対する宣言

アセンブラに対して宣言した値を合わせる！

```
        MOV.B       #03H,PRCR           ;プロテクト解除
        MOV.W       #0007H,PM0          ;プロセッサモードレジスタ 0,1 の設定
        MOV.W       #2008H,CM0          ;システムクロック制御レジスタ 0,1 の設定
        MOV.B       #0,PRCR             ;全レジスタプロテクト
;
        LDC         #0,FLG              ;フラグレジスタの初期値設定
;
;
        MOV.W       #0FFF0H,PUR1        ;内蔵プルアップ抵抗の接続
;
        MOV.W       #0,R0               ;RAM クリア
        MOV.W       #((RAM_END+1) - RAM_TOP)/2,R3
        MOV.W       #RAM_TOP,A1
        SSTR.W
;
;================ メインプログラム ====================================
MAIN:
        MOV.B       DATA_TABLE[A0],R0L
        MOV.W       #1234H,R1
        BSET        char_b0
;               •
;               •
;               •
        JMP         MAIN
;
;=============== ダミー割り込みプログラム ==============================
dummy:
        REIT
;
;================ 固定データ領域 ======================================
;
  .SECTION      CONSTANT,ROMDATA    ;セクション名、セクションタイプの宣言
;  .ORG         XXXXXH              ;開始アドレスの宣言
;
DATA_TABLE:
  .BYTE         12H,34H,56H,78H         ;1 バイトデータの設定
  .WORD         1234H,5678H             ;2 バイトデータの設定
  .ADDR         123456H,789ABCH         ;3 バイトデータの設定
  .LWORD        12345678H,9ABCDEF0H     ;4 バイトデータの設定
DATA_TABLE_END:
;
```

ハードウェア及びプログラミング
上選択した内容に合わせる

ハードウェアの ROM 領域に合わせる

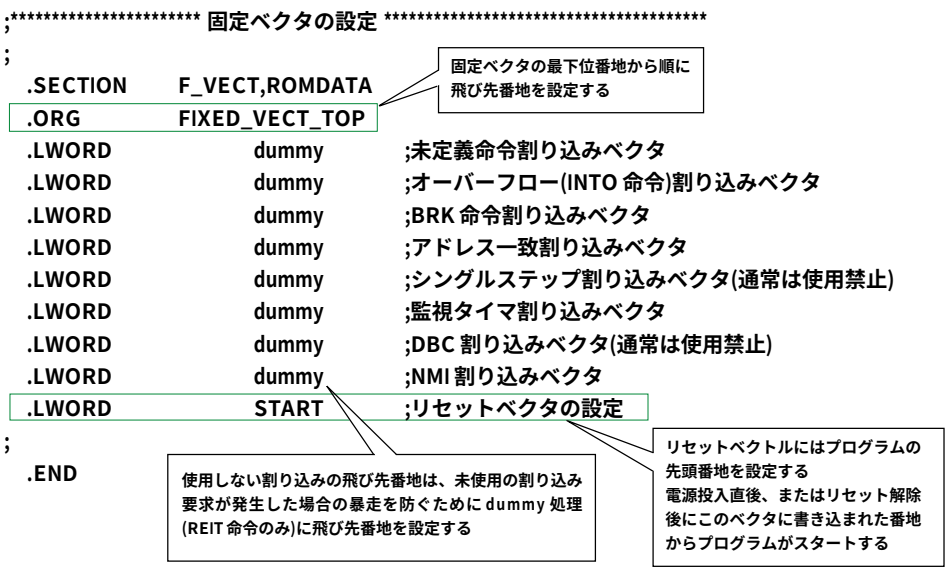


図 4.1.7　初期設定の記述例 1

4.2　CPU の初期設定

電源投入直後、またはリセット直後に各レジスタ、RAM 等の初期設定が必要です。CPU 内部のレジスタの未設定、及びプログラム実行前のメモリ等に意図しないデータが残っていた場合、プログラム暴走の原因になります。したがって初期設定はプログラムの最初におこないます。初期設定では以下の事項について設定します。

- **アセンブラに対する宣言**
- **CPU 内部のレジスタ、フラグ、RAM 領域の初期化**
- **ワーク領域の初期化**
- **ポート、タイマ、割り込み等の内蔵周辺機能の初期化**

4.2.1　CPU 内部レジスタの設定

通常はリセット解除後プロセッサのモードやシステムのクロックに関するレジスタの設定を行なう必要があります。設定例は「4.2.7　サンプルリスト 2(初期設定 2)」をご参照ください。

4.2.2　スタックポインタの設定

サブルーチンや割り込みを使用する場合、戻り先番地などがスタックに退避されます。よってサブルーチンコール前、あるいは割り込み許可前にスタックポインタの設定をおこなっておく必要があります。設定例は「4.2.7　サンプルリスト 2(初期設定 2)」をご参照ください。

4.2.3　ベースレジスタ（SB,FB）の設定

M16C/60シリーズ、M16C/20シリーズでは効率のよいデータのアクセスを行なうためにベースレジスタ相対アドレッシングというアドレッシングモードがあります。これはベースとなるアドレスからの相対番地でアクセスしますので、このアドレッシングモードを使用するにはベースアドレスをあらかじめ設定しておく必要があります。設定例は「4.2.7　サンプルリスト 2(初期設定 2)」をご参照ください。

4.2.4　割り込みテーブルレジスタ(INTB)の設定

M16C/60シリーズ、M16C/20シリーズでは割り込みベクタテーブルが可変になっているため割り込みを使用する前にベクタの先頭アドレスを設定する必要があります。設定例は「4.2.7　サンプルリスト 2(初期設定 2)」をご参照ください。

4.2.5　可変／固定ベクタの設定

M16C/60 シリーズ、M16C/20 シリーズには可変ベクタと固定ベクタの 2 種類のベクタが存在します。割り込みを使用する場合のベクタの設定方法、あるいは使用しない場合の暴走対策処理方法などについては「4.2.7　サンプルリスト 2(初期設定 2)」をご参照ください。

4.2.6　周辺機能の設定

M16C/60 グループが内蔵している RAM、ポート、タイマの初期設定について説明します。詳しくはご使用品種のユーザーズマニュアルの機能説明をご参照ください。

ワーク領域の初期設定

通常ワーク領域は初期設定で 0 クリアします。初期値が 0 でない場合、各ワーク領域に初期値を設定してください。図 4.2.1 にワーク領域の初期設定例を示します。

```
;---------------------- ワーク RAM の初期設定 -------------------------------------
;
            MOV.B       #0FFH,char
;
            MOV.W       #0FFFFH,short
;
            MOV.W       #0FFFFH,addr
            MOV.B       #0FFH,addr+2
;
            MOV.W       #0FFFFH,long
            MOV.W       #0FFFFH,long+2
;
```

図 4.2.1　ワーク領域の初期設定例

ポートの初期設定

各ポートの方向レジスタを出力に設定した時点で、ポートからデータが出力されます。不定なデータを出力しないため、方向レジスタを出力に設定する前に出力ポートに初期値を設定してください。図4.2.2にポートの初期設定例を示します。

```
;---------------------- ポートの初期値設定 ---------------------------------------
;
            MOV.W       #0FFFFH,P6          ;ポート P6,P7 初期値設定
            MOV.W       #0FFFFH,PD6         ;ポート P6,P7 を出力に設定
            MOV.B       #04H,PRCR           ;プロテクト解除
            MOV.W       #0000H,PD8          ;ポート P8,P9 を入力に設定
;
```

図 4.2.2　ポートの初期設定例

タイマの設定

タイマ等の内蔵周辺機能を使用する場合、関連レジスタ(SFR 領域)に対する初期設定　を行います。図4.2.3にタイマ A0 の設定例を示します。

```
;---------------------- タイマ A0 の設定 -----------------------------------------
;
TA0S        .BTEQU              0,TABSR

            MOV.B       #01000000B,TA0MR    ;タイマ A0 モードレジスタ設定
                                            ;(モード:タイマモード , 分周比:1/8)
            MOV.B       #00000111B,TA0IC    ;タイマ A0 割り込み要求ビットクリア
                                            ;タイマ A0 割り込み許可(優先レベル:7)
            MOV.W       #2500-1,TA0         ;タイマ A0 カウント値設定
;
            BSET        TA0S                ;タイマ A0 カウントスタート
```

図 4.2.3　タイマの設定例

4.2.7 サンプルリスト 2(初期設定 2)

```
;********************** インクルード ***********************************************
;
  .INCLUDE              m30600.inc
;
************************ シンボル定義 ************************************************
;
RAM_TOP                 .EQU   00400H   ; RAM の先頭アドレス
RAM_END                 .EQU   02BFFH   ; RAM の終了アドレス
ROM_TOP                 .EQU   0F0000H   ; ROM の先頭アドレス
FIXED_VECT_TOP          .EQU   0FFFDCH  ;固定ベクタの先頭アドレス
SB_BASE                 .EQU   00380H   ; SB 相対のベースアドレス
FB_BASE                 .EQU   00480H   ; FB 相対のベースアドレス
;
;************************ ワーク RAM 領域確保 ***************************************
;
  .SECTION      WORK,DATA
  .ORG          RAM_TOP
;
WORKRAM_TOP:
WORK_1:         .BLKB           1
WORK_2:         .BLKB           1
WORKRAM_END:
;
;********************** プログラム領域 ********************************************
;================== スタートアップ ======================================
;
  .SECTION      PROGRAM,CODE        ;セクション名、セクションタイプの宣言
  .ORG          ROM_TOP             ;開始アドレスの宣言
  .SB           SB_BASE             ; SB レジスタの値をアセンブラに対して宣言
  .FB           FB_BASE             ; FB レジスタの値をアセンブラに対して宣言
;
START:
        LDC             #RAM_END+1,ISP      ;スタックポインタの初期値設定
        LDC             #SB_BASE,SB         ; SB レジスタ初期値設定
        LDC             #FB_BASE,FB         ; FB レジスタ初期値設定
;
        MOV.B           #03H,PRCR           ;プロテクト解除
        MOV.W           #0007H,PM0          ;プロセッサモードレジスタ 0,1 の設定
        MOV.W           #2008H,CM0          ;システムクロック制御レジスタ 0,1 の設定
        MOV.B           #0,PRCR             ;全レジスタプロテクト
;
        LDC             #0,FLG              ;フラグレジスタの初期値設定
        LDINTB          #VECT_TOP           ;割り込みテーブルレジスタの初期値設定
;
        MOV.W           #0FFF0H,PUR1        ;内蔵プルアップ抵抗の接続
;
```

```
        MOV.W   #0,R0                   ;WORK_RAM 0 クリア
        MOV.W   #(RAM_END - RAM_TOP)/2,R3
        MOV.W   #WORKRAM_TOP,A1
        SSTR.W
;
;================ メインプログラム ===================================
MAIN:
        JSR     INIT    ;ワーク RAM の初期値設定
        FSET    I       ;割り込み許可
MAIN_10:
        MOV.B   WORK_1,R0L
;               •
;               •
;               •
        JMP     MAIN_10
;
;=============== INIT ルーチン ================================
INIT:
        MOV.B   #0FFH,WORK_1
        MOV.B   #0FFH,WORK_2
        MOV.B   #00000111B,TA0IC    ;割り込み要求ビットのクリア
                                    ;タイマ A0 割り込み許可(優先レベル:7)
        MOV.B   #01000000B,TA0MR    ;タイマ A0 モードレジスタ設定
        MOV.W   #2500-1,TA0         ;タイマ A0 カウント値設定
        BSET    0,TABSR             ;タイマ A0 カウントスタート
INIT_END:
        RTS
;
;=============== TA0 割り込み処理プログラム ==============================
INT_TA0:
        PUSHM   R0,R1,R2,R3,A0,A1
;               •
;               •
;               プログラム
;               •
;               •
        POPM    R0,R1,R2,R3,A0,A1
INT_TA0_END:
        REIT
;
;=============== ダミー割り込みプログラム ==============================
dummy:
        REIT
;
```

```
;********************** 可変ベクタテーブルの設定 ***************************************
;
  .SECTION      VECT,ROMDATA
  .ORG          VECT_TOP+(11*4)
;
  .LWORD              dummy       ;DMA0 割り込みベクタ
  .LWORD              dummy       ;DMA1 割り込みベクタ
  .LWORD              dummy       ;キー入力割り込みベクタ
  .LWORD              dummy       ;A-D 割り込みベクタ
  .LWORD              dummy       ;未使用
  .LWORD              dummy       ;未使用
  .LWORD              dummy       ;UART0 送信割り込みベクタ
  .LWORD              dummy       ;UART0 受信割り込みベクタ
  .LWORD              dummy       ;UART1 送信割り込みベクタ
  .LWORD              dummy       ;UART1 受信割り込みベクタ
  .LWORD              INT_TA0     ;タイマ A0 割り込みベクタに飛び先番地設定
  .LWORD              dummy       ;タイマ A1 割り込みベクタ
  .LWORD              dummy       ;タイマ A2 割り込みベクタ
  .LWORD              dummy       ;タイマ A3 割り込みベクタ
  .LWORD              dummy       ;タイマ A4 割り込みベクタ
  .LWORD              dummy       ;タイマ B0 割り込みベクタ
  .LWORD              dummy       ;タイマ B1 割り込みベクタ
  .LWORD              dummy       ;タイマ B2 割り込みベクタ
  .LWORD              dummy       ;INT0 割り込みベクタ
  .LWORD              dummy       ;INT1 割り込みベクタ
  .LWORD              dummy       ;INT2 割り込みベクタ
;
;********************** 固定ベクタの設定 ***************************************
;
  .SECTION      F_VECT,ROMDATA
  .ORG          FIXED_VECT_TOP
;
  .LWORD              dummy       ;未定義命令割り込みベクタ
  .LWORD              dummy       ;オーバフロー(INTO 命令)割り込みベクタ
  .LWORD              dummy       ;BRK 命令割り込みベクタ
  .LWORD              dummy       ;アドレス一致割り込みベクタ
  .LWORD              dummy       ;シングルステップ割り込みベクタ(通常は使用禁止)
  .LWORD              dummy       ;監視タイマ割り込みベクタ
  .LWORD              dummy       ;DBC 割り込みベクタ(通常は使用禁止)
  .LWORD              dummy       ;NMI 割り込みベクタ
  .LWORD              START       ;リセットベクタの設定
;
  .END
```

図 4.2.4　初期設定の記述例 2

4.3 割り込みの設定

ここでは割り込み処理プログラムを実行する場合に必要な処理及び記述方法、多重割り込みの実行方法に関する説明をします。

割り込み処理プログラムを実行する場合、以下の処理が必要になります。

(1) 割り込みテーブルレジスタの設定
(2) 可変／固定ベクタの設定
(3) 割り込み許可フラグの許可
(4) 割り込み制御レジスタの設定
(5) 割り込み処理ルーチン内でのレジスタの退避と復帰

4.3.1 割り込みテーブルレジスタの設定

割り込みテーブルレジスタ(INTB)によって可変ベクタの先頭番地を指定することができます。割り込みテーブルレジスタで指定された番地から256バイトが可変ベクタ領域になり、1ベクタ4バイトとなります。各ベクタにはソフトウェア割り込み番号が割り付けられており、0～63となります。

4.3.2　可変／固定ベクタの設定

割り込みが発生すると割り込み要因ごとに設定された番地にジャンプします。この番地を割り込みベクタと呼びます。

割り込みベクタの設定は、各割り込み処理プログラムの先頭番地を可変／固定ベクタテーブルに登録します。実際の登録例として「4.3.6　サンプルリスト 3(ソフトウェア割り込み)」をご参照ください。

可変ベクタテーブル

可変ベクタテーブルは、割り込みテーブルレジスタ(INTB)によって指し示された値を先頭アドレスとする 256 バイトの割り込みベクタテーブルで、ベクタテーブルは全メモリ空間のどこにでも配置が可能です。また、1 ベクタは 4 バイトで構成され、ベクタごとに 0 ～ 63 までのソフトウェア割り込み番号が割り付けられます。

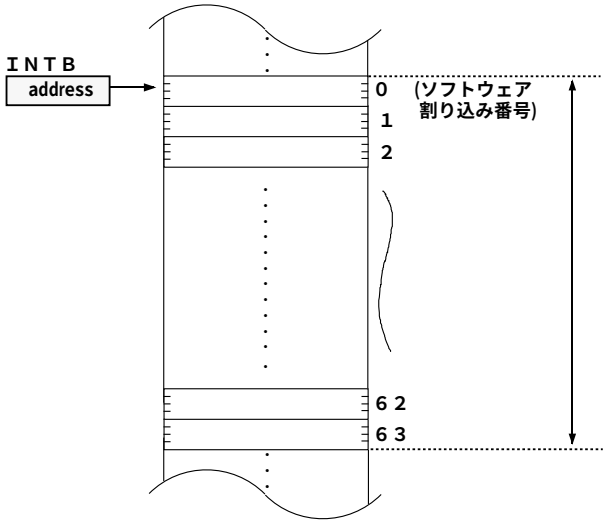


図 4.3.1　可変ベクタテーブル

4.3.3 割り込み許可フラグの許可

電源投入直後、リセット解除後は割り込み禁止状態になっていますので、プログラム中で許可にする必要があります。フラグレジスタの I フラグを "1" にセットすることで割り込み許可状態にすることができます。

なお、I フラグを "1" にセットした瞬間、割り込みが許可されます。プログラムの先頭で許可すると暴走の原因となりますので、初期設定を行った後に許可してください。

4.3.4 割り込み制御レジスタの設定

各割り込み制御レジスタのビット 0 ～ビット 2 で、各割り込みの割り込み優先レベルを設定することができます。レベル =0 の時は割り込み禁止と同じ状態になりますので、"1" 以上に設定してください。　また、割り込み制御レジスタのビット 3 が割り込み要求フラグになります。リセット解除後は "0" になりますが、"1" にセットされている可能性もありますので、割り込み許可フラグ(I フラグ)を許可する前に "0" にクリアしてください。

各割り込み制御レジスタのビット構成や優先レベル等についてはご使用品種のユーザーズマニュアルをご参照ください。

4.3.5　割り込み処理ルーチン内でのレジスタの退避と復帰

割り込みが受け付けられると、以下のものが自動的にスタックに退避されます。退避順序、スタックの動作などの詳細については「4.5.2　スタック領域」をご参照ください。

- **PC(プログラムカウンタ)**
- **FLG(フラグレジスタ)**

割り込み処理プログラムからの復帰は、必ず REIT 命令で行ってください。この命令によって割り込み処理終了後、スタックに退避されていたレジスタ、戻り先番地などが復帰され、割り込み発生前の処理を継続します。

自動的に退避されるレジスタ以外で、割り込み処理ルーチン内で使用するためそれ以前の内容を保持しておきたいレジスタについては、ソフトウェアによってスタックに退避します。割り込み処理ルーチン内でのレジスタの退避／復帰方法の例として「4.3.6　サンプルリスト 3(ソフトウェア割り込み)」をご参照ください。

レジスタの退避と復帰の種類

自動的に退避されるレジスタ以外で、割り込み処理ルーチン内で使用するためそれ以前の内容を保持しておきたいレジスタは、ソフトウェアによってスタック領域に退避します。レジスタの退避、復帰方法は 2 通りあります。以下にその処理手順を示します。

(1)プッシュ／ポップ命令による退避と復帰

(1a)個別退避

```
PUSH.B    R0L
PUSH.W    R1
```

(1b)個別復帰

```
POP.B     R0L
POP.W     R1
```

(2a)一括退避

```
PUSHM     R0,R1,R2,R3,A0,A1
```

(2b)一括復帰

```
POPM      R0,R1,R2,R3,A0,A1
```

(2)レジスタバンク切り替えによる退避と復帰

割り込み処理のオーバーヘッドタイムを短くしたい場合に有効です。

(a)レジスタバンク 1 を使用

```
FSET  B
```

(b)レジスタバンク 0 を使用

```
FCLR  B
```

割り込み処理プログラムの記述

図 4.3.2 に割り込み処理プログラムの記述例を示します。

```
****************** レジスタの個別退避と復帰 ************************************
INT_A0:
        PUSH.B    R0L                ;R0L を退避
        PUSH.B    R1L                ;R1L を退避
        PUSH.W    R2                 ;R2 を退避
        •
        •
        割り込み処理
        •
        •
        POP.W     R2                 ;R2 を復帰
        POP.B     R1L                ;R1L を復帰
        POP.B     R0L                ;R0L を復帰
        REIT                         ;割り込みからの復帰
;
```

個別で退避した場合は、復帰するときに退避した時と逆の順番で復帰させること!!

```
****************** レジスタの一括退避と復帰 ************************************
INT_A1:
        PUSHM     R0,R1,R2,R3        ;R0,R1,R2,R3 レジスタを一括して退避
        •
        •
        割り込み処理
        •
        •
        POPM      R0,R1,R2,R3        ;R0,R1,R2,R3 レジスタを一括して復帰
        REIT                         ;割り込みからの復帰
;
```

```
****************** レジスタバンク切り替えによるレジスタの退避と復帰 ********************
INT_A2:
        FSET   B                     ;レジスタバンク = 1
        •
        •
        割り込み処理
        •
        •
        FCLR   B                     ;レジスタバンク = 0
        REIT                         ;割り込みからの復帰
;
```

割り込みプログラム内では、バンク 1 の各レジスタを使用することになる (R0,R1,R2,R3,A0,A1,FB)

図 4.3.2　割り込み処理でのレジスタ復帰 / 退避

(注)メインプログラム内でレジスタバンク0、1両方を使用している場合はレジスタバンク切り替えによるレジスタの退避と復帰はできません。

4.3.6 サンプルリスト 3(ソフトウェア割り込み)

INTO命令(オーバフロー)割り込みはオーバフローフラグが "1"にセットされている状態で実行すると割り込みが発生するソフトウェア割り込みです。図 4.3.3 にソフトウェア割り込みの使用例を示します。

```
;********************** インクルード ************************************************
;
  .INCLUDE      m30600.inc
;
********************** シンボル定義 ************************************************
;
RAM_TOP             .EQU   00400H       ;RAM の先頭アドレス
RAM_END             .EQU   02BFFH       ;RAM の終了アドレス
ROM_TOP             .EQU   0F0000H      ;ROM の先頭アドレス
VECT_TOP            .EQU   0FFF00H      ;可変ベクタの先頭アドレス
FIXED_VECT_TOP      .EQU   0FFFDCH      ;固定ベクタの先頭アドレ
SB_BASE             .EQU   00380H       ;SB 相対のベースアドレス
FB_BASE             .EQU   00480H       ;FB 相対のベースアドレス
;
;********************** ワーク RAM 領域確保 ***************************************
;
  .SECTION      WORK,DATA
  .ORG          RAM_TOP
;
WORKRAM_TOP:
WORK_1:.BLKW        1
WORK_2:.BLKB        1
ANS_L:          .BLKW       1
ANS_H:          .BLKW       1
WORKRAM_END:
;
;********************** プログラム領域 ********************************************
;==================== スタートアップ ========================================
;
  .SECTION      PROGRAM,CODE
  .ORG          ROM_TOP
  .SB           SB_BASE                 ;SB レジスタの値をアセンブラに対して宣言
  .FB           FB_BASE                 ;FB レジスタの値をアセンブラに対して宣言
;
START:
        LDC         #RAM_END+1,ISP          ;スタックポインタの初期値設定
        LDC         #SB_BASE,SB             ;SB レジスタ初期値設定
        LDC         #FB_BASE,FB             ;FB レジスタ初期値設定
;
        MOV.B       #03H,PRCR               ;プロテクト解除
        MOV.W       #0087H,PM0              ;プロセッサモードレジスタ 0,1 の設定
        MOV.W       #2008H,CM0          ;システムクロック制御レジスタ 0,1 の設定
        MOV.B       #0,PRCR                 ;全レジスタプロテクト
```

```
;
        LDC     #0,FLG                  ;フラグレジスタの初期値設定
        LDINTB  #VECT_TOP       ;割り込みテーブルレジスタの初期値設定
;
        MOV.W   #0FFF0H,PUR1            ;内蔵プルアップ抵抗の接続
;
        MOV.W   #0,R0                   ;WORK_RAM 0 クリア↓
        MOV.W   #((RAM_END+1) - RAM_TOP)/2,R3
        MOV.W   #WORKRAM_TOP,A1
        SSTR.W
;
;================== メインプログラム ==============================
MAIN:
        JSR     INIT                    ;ワーク RAM の初期値設定
MAIN_10:
        MOV.W   WORK_1,R0
        DIV.B   #4                      ;符号付き除算
        INTO                            ;除算の結果オーバフローが発生した場合;
;       ——                              ;(O フラグ =1)は INTO 命令を実行し、
        MOV.B   R0L,WORK_2              ;割り込みが発生する。
;               •
;               •
;
        MOV.W   #0,R0
        MOV.W   #0,R2
        MOV.W   #1234H,A0
        MOV.W   #5678H,A1
        MOV.W   #0FFH,R3
        RMPA.W                          ;積和演算
        INTO                            ;積和演算の結果オーバフローが発生した場合
;       ——                              ;(O フラグ =1)は INTO 命令を実行し、
                                        ;割り込みが発生する。
        MOV.W   R2,ANS_H
        MOV.W   R0,ANS_L
;               •
;               •
        JMP     MAIN_10
;
;================ INIT ルーチン ============================================
INIT:
        MOV.W   #0FFFFH,WORK_1
        MOV.B   #0FFH,WORK_2
        MOV.W   #0,ANS_L
        MOV.W   #0,ANS_H
INIT_END:
        RTS
;
```

```
;================= オーバフロー割り込み処理プログラム ========================
INT_OVER_FLOW:
        PUSHM           R0,R1,R2,R3,A0,A1
;               ・
;               ・
;               プログラム
;               ・
;               ・
        POPM        R0,R1,R2,R3,A0,A1
INT_OVER_FLOW_END:
        REIT
;
;================= ダミー割り込みプログラム ================================
dummy:
        REIT
;
;********************** 固定ベクタの設定 ***************************************
;
  .SECTION      F_VECT,ROMDATA
  .ORG          FIXED_VECT_TOP
;
  .LWORD            dummy           ;未定義命令割り込みベクタ
  .LWORD            INT_OVER_FLOW   ;オーバフロー割り込みベクタ設定
  .LWORD            dummy           ;BRK 命令割り込みベクタ
  .LWORD            dummy           ;アドレス一致割り込みベクタ
  .LWORD            dummy           ;シングルステップ割り込みベクタ
                                    ;  (通常は使用禁止)
  .LWORD            dummy           ;監視タイマ割り込みベクタ
  .LWORD            dummy           ;DBC 割り込みベクタ(通常は使用禁止)
  .LWORD            dummy           ;NMI 割り込みベクタ
  .LWORD            START           ;リセットベクタ設定
;
  .END
```

図 4.3.3　ソフトウェア割り込みの使用例

4.3.7 ISP と USP

M16C/60シリーズ、M16C/20シリーズではスタックポインタが２つ（割り込みスタックポインタ(ISP)とユーザスタックポインタ(USP)）あります。使用するスタックポインタはUフラグによって切り替えることができます。

(1)**U=0 の場合　　ISP 使用**

ISP で示されるアドレスに対してレジスタの退避／復帰が行われます。

(2)**U=1 の場合　　USP 使用**

USP で示されるアドレスに対してレジスタの退避／復帰が行われます。

アセンブリ言語でのみプログラミングする場合(OSを使用しない場合)はISPを使用してください。USPを使用することもできますが、周辺I/O割り込みの使用において注意が必要です。詳しくは同項の"ソフトウェア割り込み番号とスタックポインタの関係"をご参照ください。

ソフトウェア割り込み番号の割り当てについて

M16C/60 シリーズ、M16C/20 シリーズではソフトウェア割り込み番号が 0 ～ 63 まであります。11 ～ 31 番は周辺 I/O 割り込みで予約されています。したがって、残りの 0 ～ 10,32 ～ 63 番にソフトウェア割り込み(INT 命令)を割り当ててください。

ただし応用上、ソフトウェア割り込み番号 32 ～ 63 番の割り込みは OS(リアルタイムモニタ MR30)等で使用するソフトウェア割り込みとして割り当てられます。基本的にはソフトウェア割り込み番号 0 ～ 10 番を使用してください。

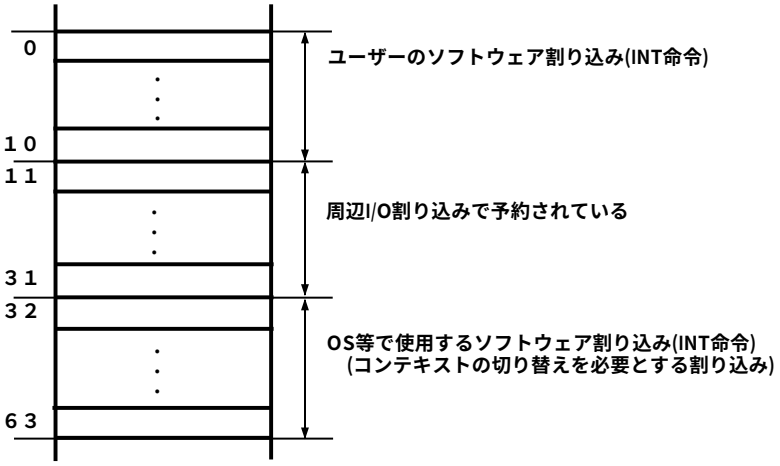


図 4.3.4　割り込み番号の割り当て

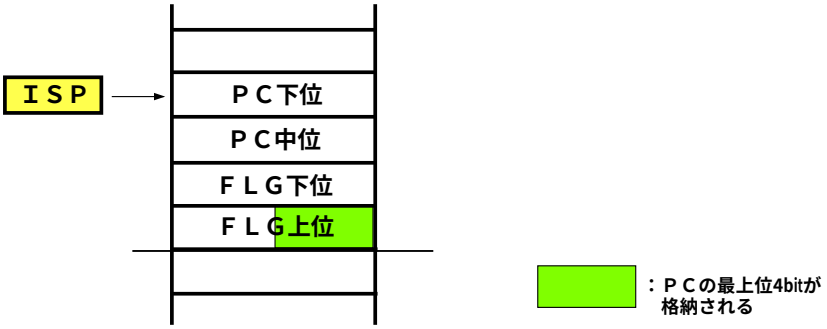
(注)OSを使用しない場合、32～63番にソフトウェア割り込み等を割り当てることもできます。この場合、スタックポインタの設定には注意が必要です。

ソフトウェア割り込み番号とスタックポインタの関係

(1)ソフトウェア割り込み番号 0 ～ 31 番の割り込み発生時

① FLG レジスタの内容を CPU 内部の一時レジスタに退避する。
② FLG レジスタの U、I、D フラグをクリアする。
②の動作により、
・スタックポインタは強制的に割り込みスタックポインタ(ISP)になる。
・多重割り込みを禁止する。
・デバッグモードをクリアする。(シングルステップを行わない。)
③ CPU 内部の一時レジスタ(FLG を退避しておいたレジスタ)、及び PC レジスタの内容をスタック領域へ退避する。
④受け付けた割り込みの割り込み要求ビットを "0" にする。
⑤プロセッサ割り込み優先レベル(IPL)に、受け付けた割り込みの割り込み優先レベルを設定する。
⑥割り込みベクタに書かれているアドレスを PC レジスタに代入する。

<割り込み要求受け付け後のスタックの状態>



<割り込み要求受け付け後のＦＬＧの状態>

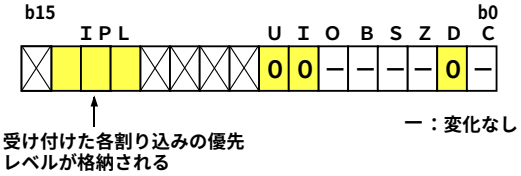


図 4.3.5　ソフトウェア割り込み番号 0 ～ 31 番割り込み発生時

(2) ソフトウェア割り込み番号 32 ～ 63 番の割り込み発生時

① FLG レジスタの内容を CPU 内部の一時レジスタに退避する。
② FLG レジスタの I、D フラグをクリアする。
②の動作により、
・スタックポインタは割り込み発生時のスタックポインタを使用する。
・多重割り込みを禁止する。
・デバッグモードをクリアする。(シングルステップを行わない。)
③ CPU 内部の一時レジスタ(FLG を退避しておいたレジスタ)、及び PC レジスタの内容をスタック領域へ退避する。
④受け付けた割り込みの割り込み要求ビットを "0" にする。
⑤プロセッサ割り込み優先レベル(IPL)に、受け付けた割り込みの割り込み優先レベルを設定する。
⑥割り込みベクタに書かれているアドレスを PC レジスタに代入する。

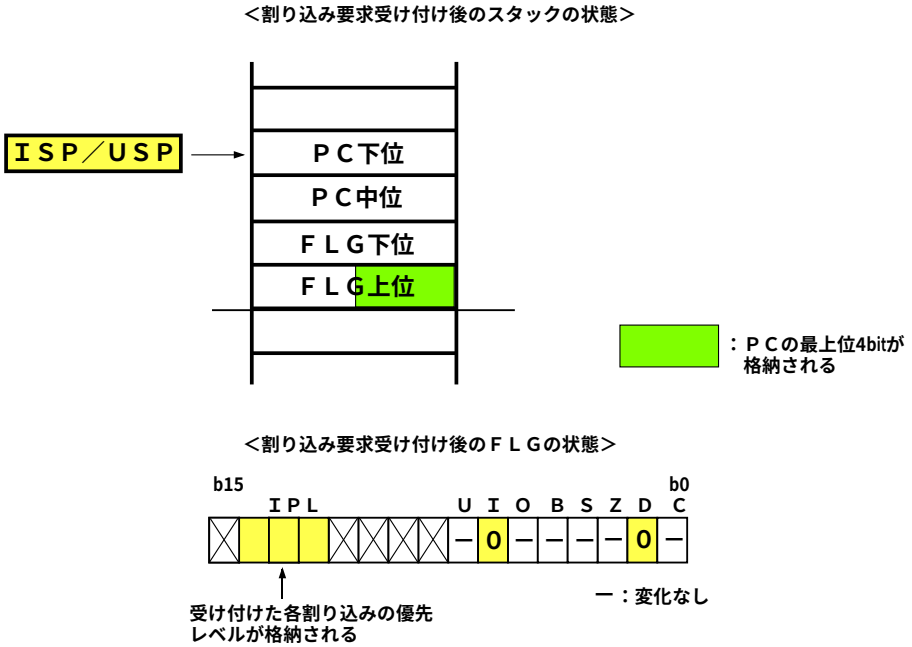


図 4.3.6　ソフトウェア割り込み番号 32 ～ 63 番割り込み発生時

注）ソフトウェアで設定した割り込み優先レベルが同じ割り込みが 1 命令実行中に複数発生した場合は、ハードウェア割り込み優先レベルに従って割り込みが発生します。
例）M16C/60 グループのハードウェア割り込み優先レベル　$\overline{\text{INT1}}$ > タイマ B2 > タイマ B0 > タイマ A3 > タイマ A1 > $\overline{\text{INT2}}$ > $\overline{\text{INT0}}$ > タイマ B1> タイマ A4 > タイマ A2 > UART1 受信 > UART0 受信 >A-D 変換 > DMA1> タイマ A0 > UART1 送信 > UART0 送信 > キー入力割り込み > DMA0

4.3.8　多重割り込み

通常の割り込み処理では１つの割り込みが許可されると割り込み許可フラグ(I)が'0'（割り込み禁止）となり、許可された割り込み処理が終了するまで他の割り込みを受け付けません。ただし、プログラム中でこの割り込み許可フラグを'1'にセット（割り込み許可）することにより多重割り込みが可能になります。

多重割り込み実行例

多重割り込みの実行例として、①②③の多重割り込みが発生した場合の実行フローを図 4.3.7 に示します。

①メインルーチン実行中、"割り込み１"が発生
②"割り込み１"実行中、"割り込み２"が発生
③"割り込み２"実行中、"割り込み３"が発生

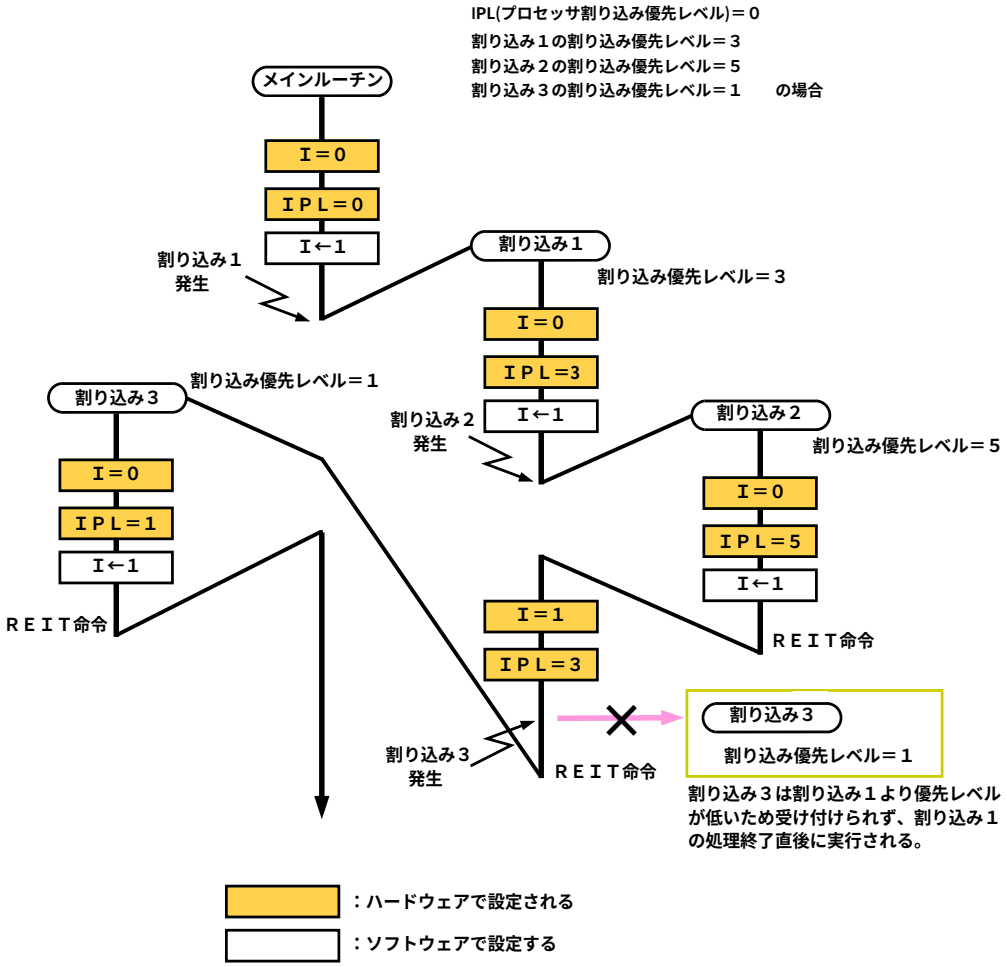


図 4.3.7　多重割り込みの実行例

4.4　ソースファイルの分割

プログラムはいくつかのソースファイルに分割して記述します。分割することによって、プログラムがまとまり読みやすくなります。さらに、ファイルごとにアセンブルできるため、修正時のアセンブル時間を短縮することができます。この節ではソースファイルの分割について説明します。

4.4.1　セクションの概念

*アセンブリ言語で書かれたプログラムは一般的にワーク領域、プログラム領域、定数データ領域から構成されます。アセンブラ(as30)でソースファイル(***.A30)をアセンブルするとリロケータブルモジュールファイル(***.R30)が生成されます。リロケータブルモジュールファイルはこのような領域を１つ以上含んでいます。セクションはこの各領域ごとに付けられた名前です。つまりセクションとはプログラムの構成要素別に付けられた名前ということになります。*

なお、アセンブラ(as30)ではアブソリュートファイルの場合でも、１つのファイルにつき必ず１つ以上のセクションを指定しなければなりません。

セクションの機能

リンク時に、同一セクション名の領域を指定されたファイル順に連続して配置します。また各セクションの開始アドレスをリンク時に指定することができます。これはつまり、各セクションの再配置がソースプログラムを変更することなく何回でも可能ということです。図 4.4.1 に実際のセクションの配置例を示します。

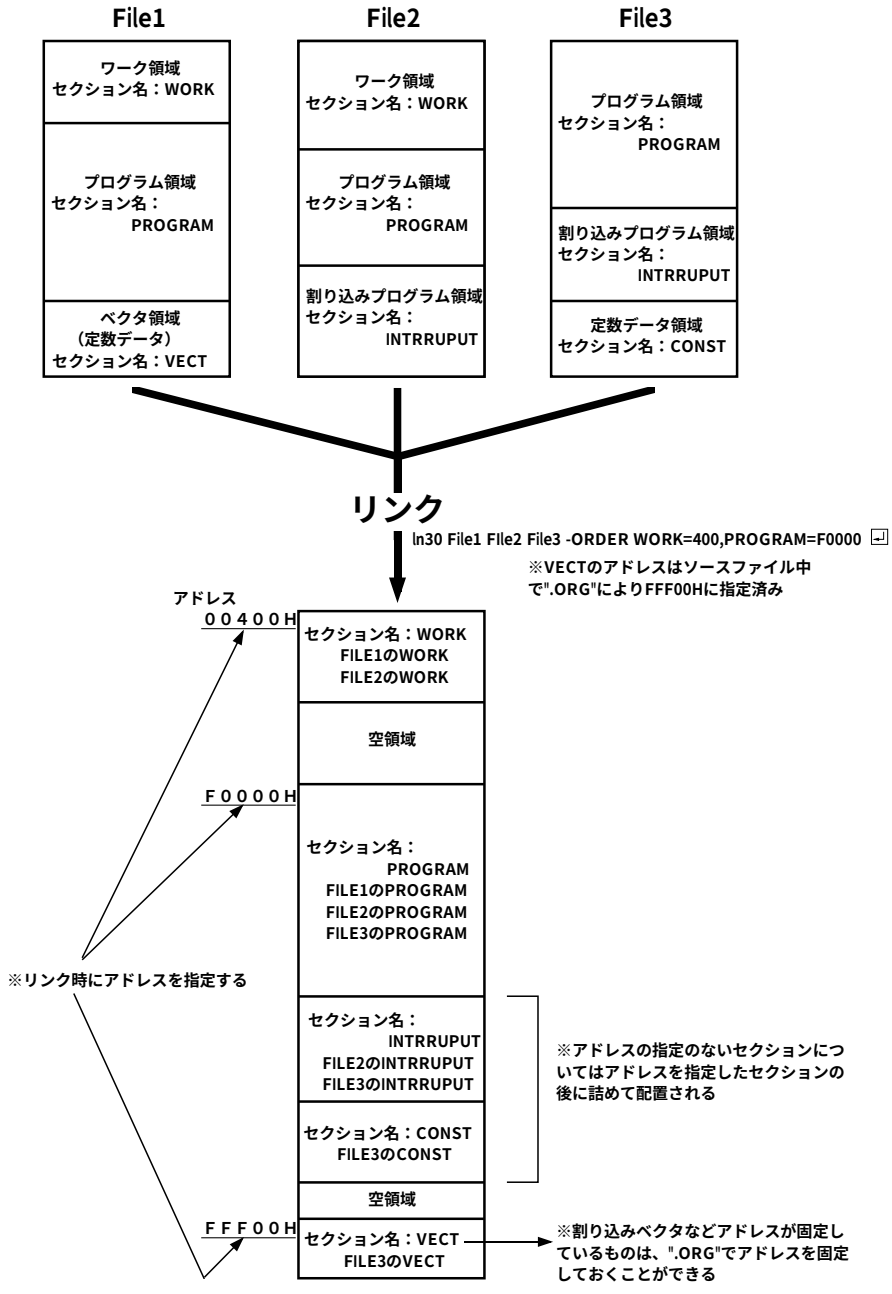


図 4.4.1　セクション配置例

4.4.2 ソースファイルの分割

本書で使用するas30はリロケータブルアセンブラです。リロケータブルアセンブラでは通常プログラムソースをいくつかのファイルに分割して記述します。ファイルを分割することによるメリットを以下に示します。

(1)プログラム、データの共有化

開発プロジェクト間のデータの交換が容易になり、既存ソフトウェアから必要な部分のみ再利用することができます。

(2)アセンブル時間の短縮

変更（修正）したファイルのみ再アセンブルするだけですみ、アセンブル時間を短縮することができます。

この項ではファイルを３つに分割(定義、メインプログラム、サブルーチン処理)して記述した場合のソースプログラムの記述例について説明します。

分割例 1　定義(WORK.A30)

ワーク RAM 領域、データテーブルの定義をファイル 1 に記述します。

```
;***********************************************************************
;                    ファイル 1（WORK.A30）
;***********************************************************************
;=================== ワーク RAM　領域確保 ================================
;
  .SECTION      WORK,DATA
  .ORG          RAM_TOP
  .GLB          WORK_1,WORK_2,WORK_3,WORK_4 ;グローバルラベルとして処理する
  .GLB          DATA_TABLE                  ;グローバルラベルとして処理する
  .BTGLB        W1_b0,W2_b1                 ;グローバルビットシンボルとして
                                            ;処理する
;
GLOBAL_WORK_TOP:
WORK_1:         .BLKB  1                    ;ワーク RAM 領域の確保↓
WORK_2:         .BLKB  1
WORK_3:         .BLKB  1
WORK_4:         .BLKB  1
GLOBAL_WORK_END:
W1_b0           .BTEQU       0,WORK_1       ;ビットシンボルの定義↓
W2_b1           .BTEQU       1,WORK_2       ;
;
;
;=================== 固定データ領域 ======================================
;
  .SECTION      CONSTANT,ROMDATA
  .ORG          CONST_TOP
;
DATA_TABLE:
          .BYTE  12H                        ;1 バイトデータの設定
          .BYTE  34H
          .BYTE  56H
          .BYTE  78H
DATA_TABLE_END:
;
  .END
```

ワーク RAM、及びラベルを別ファイルから参照可能にするため、.GLB によりグローバルラベルの宣言を行う

.BTEQU で定義されたビットシンボルを別ファイルから参照可能にするため、.BTGLB によりグローバルシンボルの宣言を行う

図 4.4.2　分割ファイル 1(WORK.A30)

分割例 2　メインプログラム(MAIN.A30)

メインプロプログラムをファイル 2 に記述します。

```
;***************************************************************************
;                    ファイル 2（MAIN.A30）
;***************************************************************************
;=================== アセンブラに対する宣言 ==============================
;
  .SECTION   PROGRAM,CODE
  .GLB    WORK_1,WORK_2,WORK_3,WORK_4;外部参照ラベルとして処理する
  .GLB    SUB_1                      ;外部参照ラベルとして処理する
  .BTGLB          W1_b0,W2_b1        ;外部参照ビットシンボルとして処理する
  .SB     00380H                     ;SB レジスタの値をアセンブラに対して設定
  .FB     00480H                     ;FB レジスタの値をアセンブラに対して設定
  .SBSYM    WORK_1,WORK_2            ;指定したラベルを SB 相対アドレッシング
                                     ;モードでコード化する
  .FBSYM    WORK_3,WORK_4            ;指定したラベルを FB 相対アドレッシング
                                     ;モードでコード化する
  .OPTJ   JSRW                       ;最適化対象外のサブルーチンコール命令
                                     ;を "JSR.W" で生成します。
;
;=================== プログラム領域 ======================================
MAIN:
          LDC     #380H,SB           ;SB レジスタ初期値設定
          LDC     #480H,FB           ;FB レジスタ初期値設定

          MOV.B WORK_1,WORK_2        ;各ワーク RAM を外部参照↓
          MOV.B WORK_3,WORK_4
;
          BSET   W1_b0               ;各ビットシンボルを外部参照↓
          BCLR   W2_b1
;
          JSR     SUB_1              ;ファイル 3 内、SUB1 をコール
;           •
;           •
;           •
  .END
```

別ファイルで定義されているラベルのため、.GLB による外部参照指定を行う

別ファイルでシンボル定義されているため、.BTGLB による外部参照指定を行う

外部参照シンボルのため、ベースレジスタ相対アドレッシングの範囲内にあるかはアセンブル時に判断できない。
よって、.SBSYM、または .FBSYM により強制的にベースレジスタ相対アドレッシングコード化する。

SB 相対アドレッシングでアクセスを行う

FB 相対アドレッシングでアクセスを行う

別ファイルのサブルーチン(ラベル)をコール(ジャンプ)する場合、アドレスが未確定のため通常は JSR.A で全てコード化される(飛び先のアドレス計算により JSR 命令の最適化がおこなえないため)。
よって、.OPTJ により JSR 命令を全て JSR.W にコード化させる。
注意)JSRW または JMPW に指定する場合は、コール(ジャンプ)命令が存在するアドレスから 64K バイト以内にサブルーチン(ラベル)があることを確認した上で行うこと!!

JSR.W にコード化され、PC 相対で分岐する

図 4.4.3　分割ファイル 2(MAIN.A30)

分割例 3　サブルーチン処理(SUB_1.A30)

サブルーチン処理をファイル 3 に記述します。

```
;***************************************************************************
;                    ファイル 3（SUB_1.A30）
;***************************************************************
;********************** ワーク RAM 領域確保 ********************
;
  .SECTION      WORK,DATA
;
LOCAL_WORK_TOP:
LOCAL_1:              .BLKB       1       ;ローカルデータの領域確保
LOCAL_2:              .BLKB       1
LOCAL_WORK_END:
;
;********************** アセンブラに対する宣言 ******************
;
  .SECTION      PROGRAM,CODE
  .GLB          SUB_1                     ;グローバルラベルとして処理する
  .GLB          DATA_TABLE                ;外部参照ラベルとして処理する
;
  .SB           00380H                    ;SB レジスタの値をアセンブラに対して設定
  .FB           00480H                    ;FB レジスタの値をアセンブラに対して設定
  .SBSYM        LOCAL_1,LOCAL_2           ;指定したラベルを SB 相対アドレッシング
                                          ;モードでコード化する
;==================== プログラム領域 ==============
SUB_1:
        LDC     #380H,SB                  ;SB レジスタ初期値設定
        LDC     #480H,FB                  ;FB レジスタ初期値設定
;
        MOV.B #05H,LOCAL_1                ;ローカルデータ(LOCAL_1)を SB 相対
                                          ;アドレッシングでアクセス
;
        MOV.W#0,A0
        LDE.B  DATA_TABLE[A0],LOCAL_2     ;外部参照による固定データテーブルの検索
        ADD.B  LOCAL_1,LOCAL_2            ;ローカルデータ(LOCAL_1,LOCAL_2)の加算
;       ・
;       ・
;       ・
        RTS                               ;サブルーチンからの復帰

  .END
```

グローバル宣言をしなければ、ファイル 3 (SUB_1.A30)のローカルラベルとして扱われる

ファイル 2(MAIN.A30)からサブルーチン(SUB_1)をコールしているため、.GLB で SUB_1 をグローバルラベルに指定しておく(ファイル内に存在するラベルのためグローバル宣言となる)

別ファイル(ファイル 1)で定義されているラベルのため、外部参照指定を行う

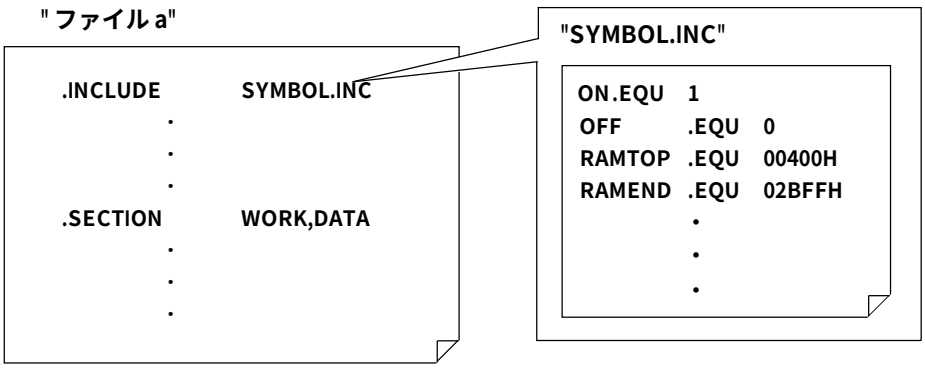
相対属性のセクションのため、リンクするまでラベルのアドレスが未定となる。したがって、.SBSYM により強制的に SB レジスタ相対アドレッシングにコード化する。
注意).SBSYM(.FBSYM)によりデータを指定する場合は、そのデータが SB/FB 相対アドレッシングの範囲内にあることを確認した上で行うこと!!

図 4.4.4　分割ファイル 3(SUB_1.A30)

インクルードファイルの利用

通常シンボルやビットシンボル(.EQU, .BTEQUで定義したもの)、及びラベル(アドレス情報を持つもの)の外部参照指定の部分を１つのインクルードファイルにします。シンボルやラベルを外部参照する場合、各ファイルごとに外部参照を指定しなくても、インクルードファイルを読み込むことにより外部参照することができます。

(1)シンボルの参照例



(2)グローバルラベルの参照例

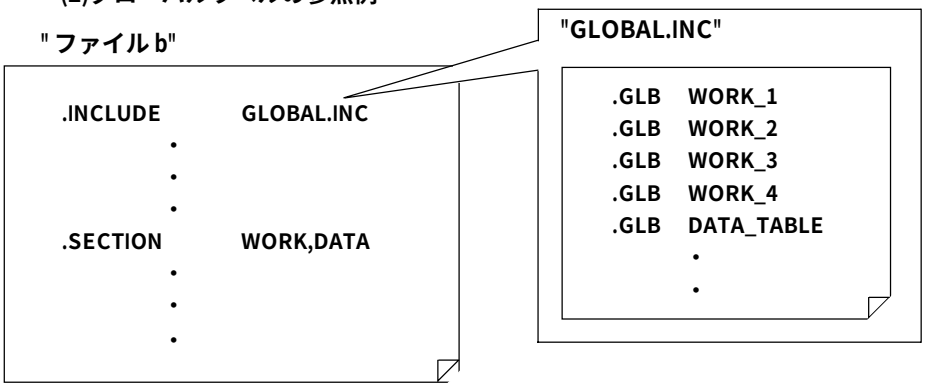


図 4.4.5　インクルードファイル例

指示命令 .LIST の利用

指示命令 ".LIST　ON, .LIST　OFF" をファイルの先頭と最後に記述することで、アセンブラリストファイルへのインクルードファイルの出力を停止することができます。図 4.4.6 にこの指示命令を使用しない場合(展開 1)と使用する場合(展開 2)のアセンブラリストファイルを示します。

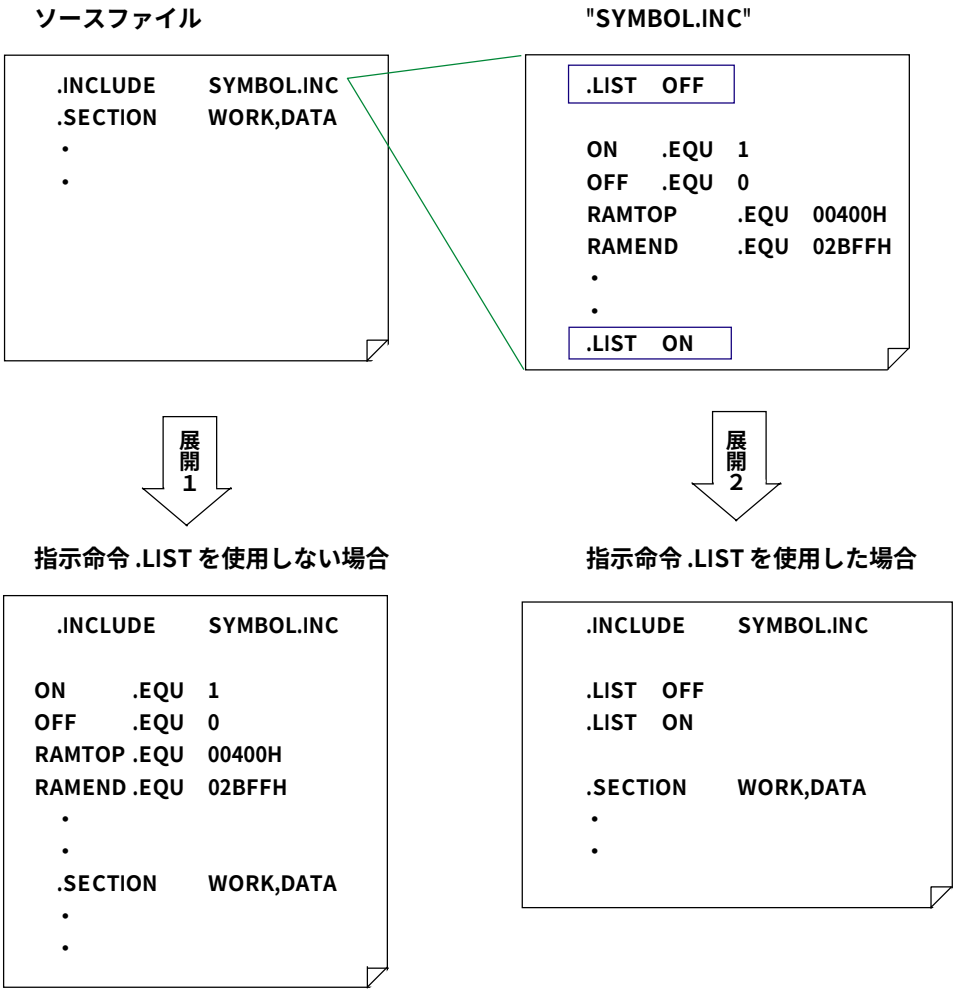


図 4.4.6　指示命令 .LIST の利用

4.4.3　ライブラリファイル

*ライブラリファイルとはいくつかのリロケータブルモジュールファイルを集めたものです。頻繁に使用するモジュールは、AS30 システム付属のライブラリアン(lib30)によって 1 つのライブラリファイルとして集めます。そのファイル(***.LIB)をリンク時に指定することで、必要なモジュールのみ（ファイル中で外部参照指定したモジュールのみ）抽出してリンクすることができます。これによりアセンブル時間の短縮、プログラムの再利用などが可能になります。以下にライブラリファイルの生成とライブラリファイルをリンクする場合の例を示します。*

ライブラリファイルの生成

ライブラリファイルの生成例を図 4.4.7 に示します。

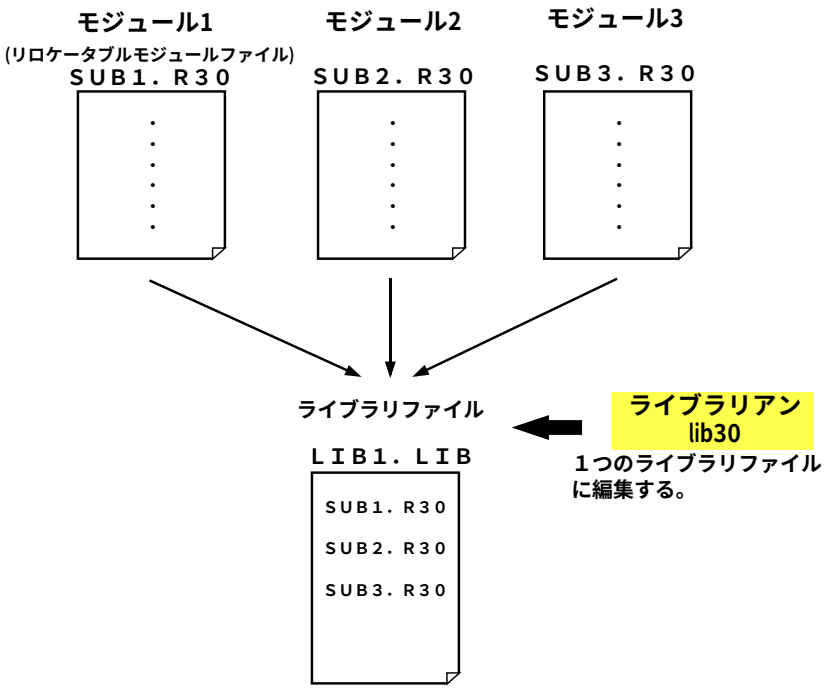


図 4.4.7　ライブラリファイルの生成

ライブラリファイルをリンクする場合の例

ライブラリファイルをリンクする場合の例を図 4.4.8 に示します。

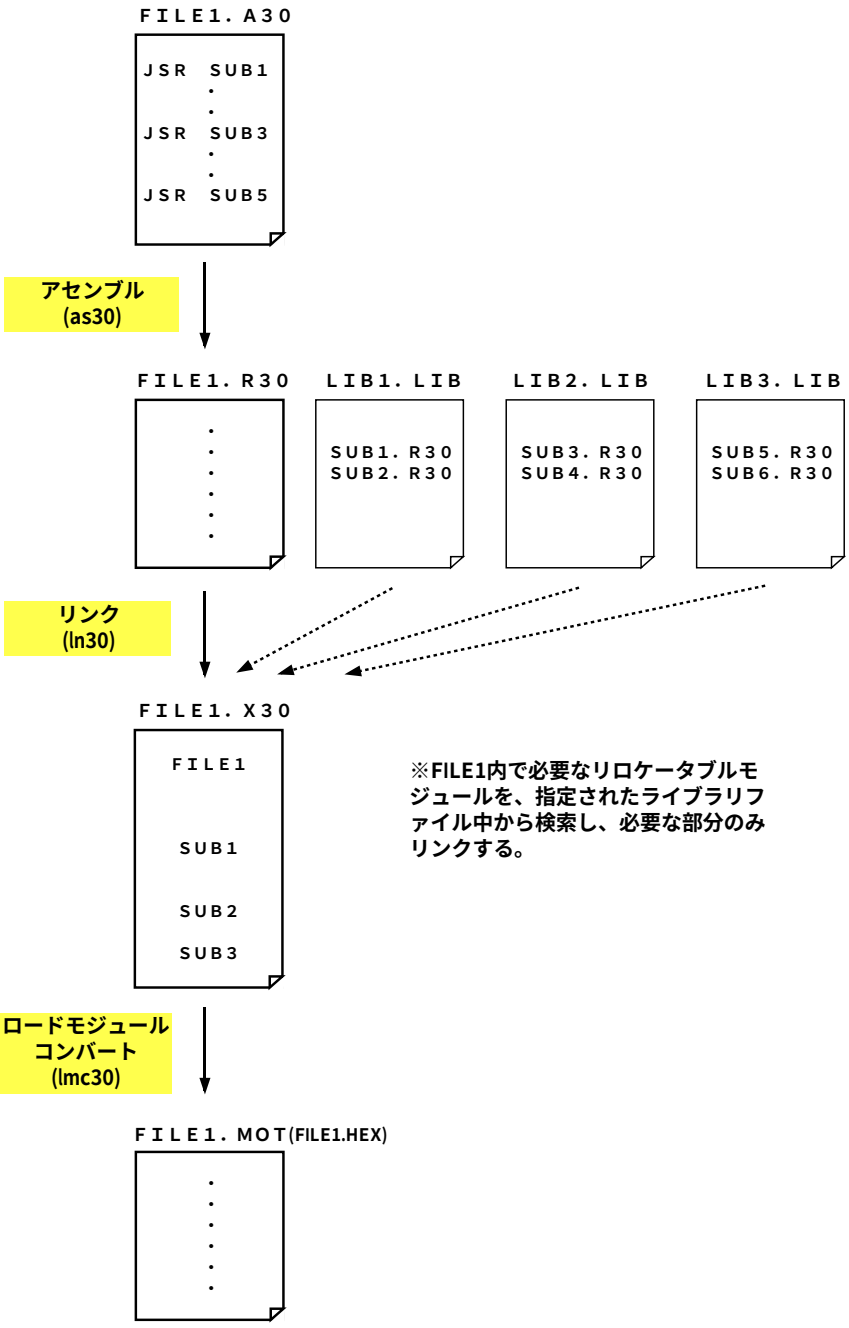


図 4.4.8　ライブラリファイルとリロケータブルモジュールファイルのリンク例

4.5　ちょっと小耳を...

この節ではM16C/60シリーズ、M16C/20シリーズを使用する上で、知っておくと便利な事項をまとめています。項目ごとにご参照ください。

4.5.1　スタック領域

スタックポインタの設定方法と、割り込みとサブルーチンを使用したときのスタック領域への退避と復帰について説明します。

スタックポインタ(ISP/USP)の設定方法

①使用するスタックポインタ(ISP,USP)の選択
通常アセンブラのみ使用する場合はISPを選択してください。詳しくは「4.3.7　ISPとUSP」をご参照ください。

②選択したスタックポインタレジスタに初期値を設定する。
M16C/60グループのスタックはFILO方式ですので、スタックポインタの初期値はRAMの最終アドレスに設定することを推奨します。

例）割り込みスタックポインタに "2C00H" を設定する場合

```
LDC    #00000000B,FLG          ;割り込みスタックポインタ(ISP)を使用
LDC    #02C00H,ISP             ;ISP に "2C00H" を設定
```

(注 1) FILO(ファースト・イン・ラスト・アウト)方式：レジスタ等の退避はアドレスの大きい方から小さい方に向かって順に積まれ、復帰する場合は最後に退避したものから順にアドレスの大きい方に向かって取り出されていく方式。
(注 2)FLG,ISP は専用レジスタです。LDC 命令(専用レジスタへの転送)を使用して設定してください。

スタック領域への退避と復帰

以下の場合にスタック領域にレジスタ等が退避／復帰されます。

(1)割り込み受け付け時

割り込みが受け付けられた場合、以下に示すレジスタがスタック領域に退避されます。

プログラムカウンタ(PC) → 下位２バイト
フラグレジスタ(FLG) → ２バイト 計４バイト

割り込み処理終了後、REIT 命令によりスタック領域に退避されていた上記のレジスタが復帰されます。

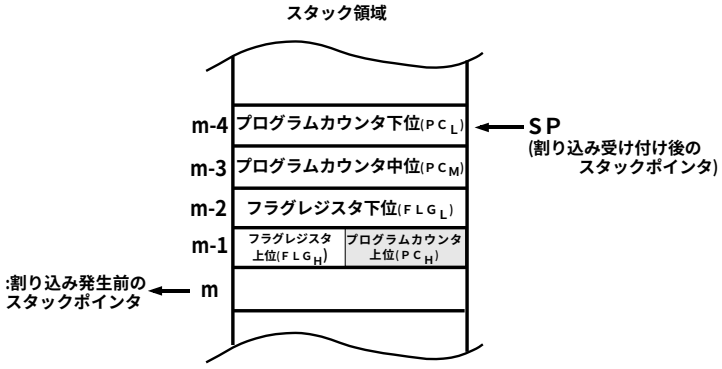


図 4.5.1 割り込み受け付け時のスタック退避／復帰

(2)サブルーチン呼び出し時(JSR、JSRI、JSRS 命令実行時)

JSR または JSRI または JSRS 命令が実行された場合、以下に示すレジスタがスタック領域に退避されます。

プログラムカウンタ(PC) → ３バイト 計３バイト

サブルーチン終了後、RTS 命令によりスタック領域に退避されていた上記のレジスタが復帰されます。

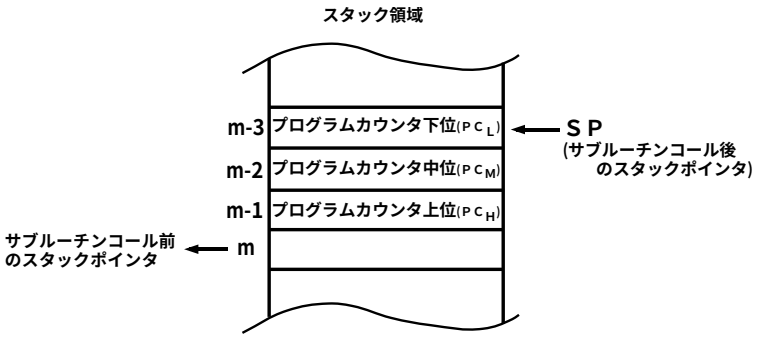


図 4.5.2 サブルーチン呼び出し時のスタック退避／復帰

4.5.2　SB、FB レジスタの設定値

この項では SB レジスタと FB レジスタの設定値について説明します。

一般的な SB、FB レジスタの設定値

頻繁にアクセスするようなデータを持つ領域の先頭アドレスをSB,FBレジスタに設定すると効果的です。よって、SFR 領域やワーク RAM 領域の先頭アドレスを設定します。

設定例を図 4.5.3 に示します。

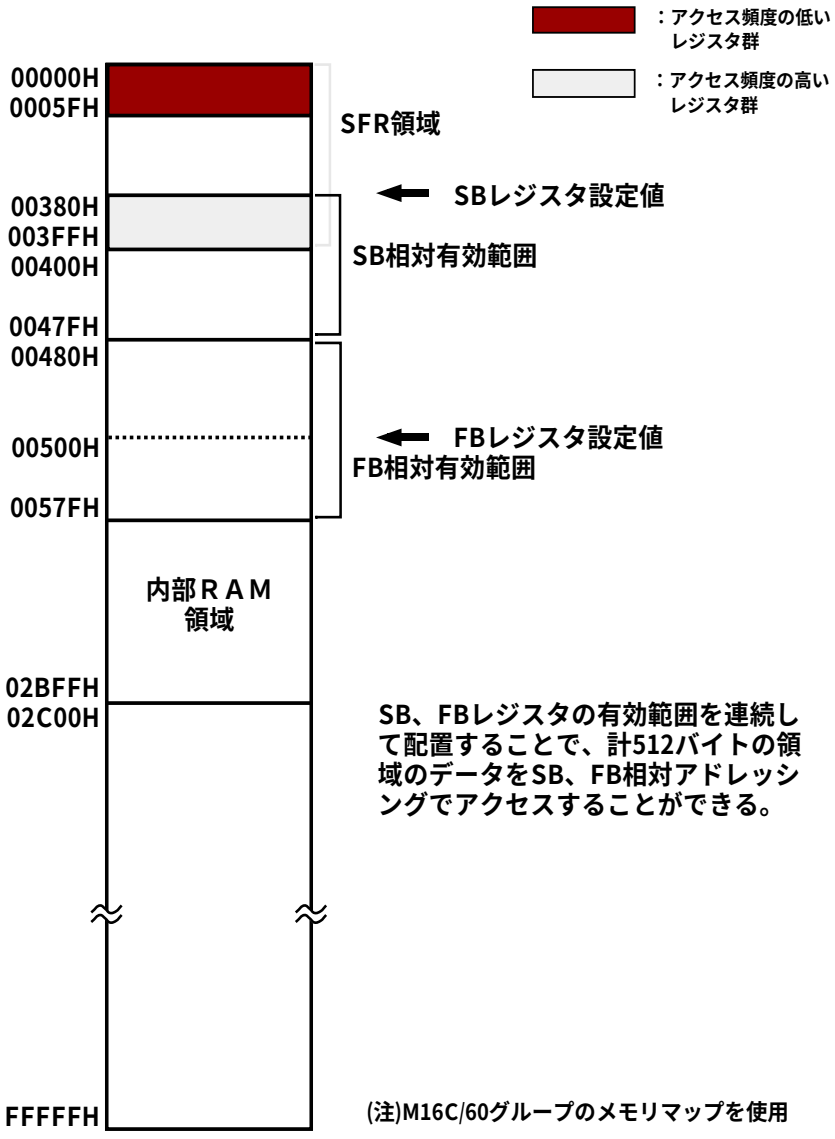


図 4.5.3　SB、FB レジスタの一般的設定値

4.5.3 アライメントの指定

この項ではアライメントの指定について説明します。

アライメントの指定とは

指示命令 ".ALIGN" を記述した直後の行のコードを格納するアドレスを偶数に補正します。セクションタイプが "CODE" または "ROMDATA" の場合は、アドレス補正した結果空になったところに NOP 命令が書き込まれます。セクションタイプが "DATA" の場合は、アドレス値を +1 します。なお、本指示命令を記述した箇所のアドレスが偶数の場合は、補正は行われません。

この指示命令は以下の条件の時に記述できます。

(1)相対属性のセクションの場合

セクション定義でアドレス補正を指示している時のみ

```
.SECTION    WORK, DATA, ALIGN
```

(2)絶対属性のセクションの場合

特に制限なし

```
.SECTION    WORK, DATA
.ORG        400H
```

アライメント指定(偶数アドレスへの補正)のメリット

データテーブルなどサイズの異なるデータが連続して配置されている場合、奇数サイズの次のデータは奇数番地に割り付けられます。M16C/60 シリーズ、M16C/20 シリーズでは偶数番地から始まるワードデータ(2 バイトのデータ)は 1 回のアクセスでリード／ライトされますが、奇数番地から始まるワードデータをアクセスする場合は 2 回のアクセスを必要とします。よって、アクセスするデータを偶数番地に配置することで、命令の実行速度を上げることができます。ただし、この場合 ROM(または RAM)効率は下がります。

図 4.5.4 にアライメント指定したプログラムの記述例を示します。

(1)セクションが相対属性の場合

```
                                    アドレス      コード
  .SECTION      WORK, DATA, ALIGN
WORK_1 .BLKW        1             00000H
WORK_2 .BLKW        1             00002H
WORK_3 .BLKB        1             00004H
  .ALIGN                          00005H        アドレスを +1
;
        .
                .                                データテーブルなどのセクションは
  .SECTION      CONST, ROMDATA, ALIGN              できるだけ偶数番地に設定する!!
  .BYTE         12H               00000H        12H
  .ALIGN                          00001H        04H     NOP コードを挿入
  .WORD         3456H             00002H        5634H
                .
                .
```

(2)セクションが絶対属性の場合

```
                                    アドレス      コード
  .SECTION      WORK, DATA
  .ORG          400H                             データテーブルなどのセクションは
WORK_1 .BLKB        1                            できるだけ偶数番地に設定する!!
  .ALIGN                          00401H        アドレスを +1
WORK_2 .BLKW        1             00402H
WORK_3 .BLKA        1             00404H
  .ALIGN                          00407H        アドレスを +1
WORK_4 .BLKL        1             00408H
;
  .SECTION      PROGRAM,CODE
  .ORG          0F0000H
        MOV.W#0,R0                F0000H        D900H
                .
                .
```

図 4.5.4　アライメント指定例

4.5.4　監視タイマ

この項では監視タイマに関する注意点と使用方法を説明します。

監視タイマとは

監視タイマは15ビットで構成されたタイマで、プログラムの暴走を検出するのに使用します。プログラムが暴走した場合、監視タイマがアンダーフローし、監視タイマ割り込みを発生します。この監視タイマ割り込み処理の中で、ソフトウェアリセット等によりプログラムを再スタートすることができます。

監視タイマ割り込みはノンマスカブル割り込みです。リセット解除後は監視タイマは停止していますが、監視タイマスタートレジスタへの書き込みによりカウントを開始します。

暴走検出方法

暴走検出時の動作フロー、および検出方法を以下に示します。

(1)動作フロー

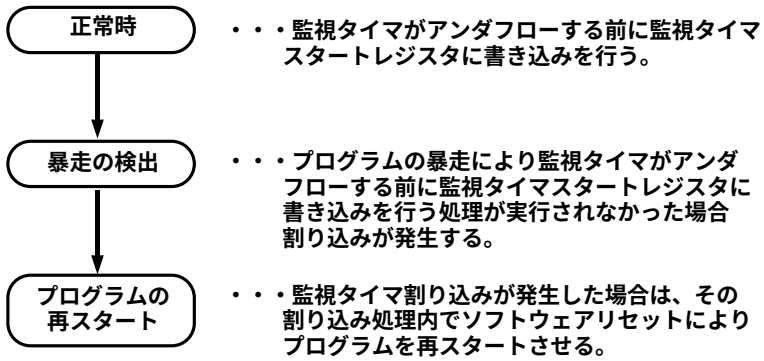


図 4.5.5　暴走検出時の動作フロー

(2)暴走の検出方法

監視タイマがアンダフローする前に監視タイマスタートレジスタに書き込みを行うようにプログラムします。監視タイマスタートレジスタに書き込みを行うことによって、監視タイマに"7FFFH"がセットされます。(任意の値は設定できません)

書き込み動作を何ヶ所にも入れた場合、プログラムが暴走したにも関わらず、暴走した先で書き込み動作が行われ暴走が検出されない場合が考えられます。したがって、書き込みを行う箇所はメインルーチンなど必ず実行される箇所に１ヶ所入れることを目標にしてください。ただしメインルーチンの長さ、または割り込み処理の長さなどを考え、監視タイマ割り込みが発生する前に監視タイマスタートレジスタに書き込みを行うようにしてください。

(3)暴走したプログラムの再スタート

割り込み処理ルーチン内で、プロセッサモードレジスタ0のビット3(ソフトウェアリセットビット)に"1"を書き込むようにプログラムします。これによりソフトウェアリセットが発生し、プログラムはリセット状態から再スタートします。（この時内部 RAM の内容はリセット直前の状態で保持されます。）

また、あらかじめ監視タイマ割り込みの割り込みベクタにこの割り込み処理プログラムの先頭番地を設定しておいてください。

なお、リセット状態から再スタートさせる時は、必ずソフトウェアリセットでリセットを行ってください。監視タイマ割り込みの割り込みベクタにリセットベクタと同じ値(アドレス)を設定した場合、IPL(プロセッサ割り込み優先レベル)が"7"のままクリアされません。したがって、リセット状態から再スタートしても他のすべての割り込みが禁止されてしまいます。

暴走検出プログラム例

図 4.5.6 と図 4.5.7 に監視タイマをプログラム暴走の検出に使用した場合のプログラム例を示します。

例 1）監視タイマスタートレジスタへの書き込み動作を(サブルーチン)、一定時間ごとに実行する

```
WDT_SET:
        MOV.B R0L,WDTS          ;監視タイマスタートレジスタへの書き込み
        RTS
```

".EQU" でアドレスを定義しておく

監視タイマスタートレジスタに任意の値は設定できないので、R0L の値は不定でよい

図 4.5.6　暴走検出プログラム例 1

例 2)監視タイマ割り込みが発生した場合、システムを再スタートする割り込み処理プログラムを実行する

```
WDT_INT:
        LDC     #00380H,SB      注 1)   ;SB,FB レジスタの再設定
        LDC     #00500H,FB
;
        BSET    1, PRCR                 ;プロセッサモードレジスタ 0、1 への書き込み許可
                                        ;(プロテクト解除)
        BSET    3, PM0                  ;ソフトウェアリセット
;
        REIT    注 2)
        •
        •
        •
  .SECTION      VECT,ROMDATA
  .ORG          0FFFF0H
  .LWORD                WDT_INT         ;監視タイマ割り込みベクタに割り込み処理の先頭番
                                        ;地を設定しておく
        •
        •
```

".EQU" でアドレスを定義しておく

プロテクトを解除してからソフトウェアリセットビットを "1" にセットし、ソフトウェアリセットをかける

注 1) 暴走時ベースレジスタ(SB,FB)の内容は保証されない。従って、SFR へ書き込む前に値を再設定する必要がある。
注 2)ソフトウェアリセットビットを "1" にセットした直後にリセットシーケンスに入る。よってそれ以降の命令は実行されない。

図 4.5.7　暴走検出プログラム例 2

4.6 参考プログラム集

この節ではM16C/60シリーズ、M16C/20シリーズのプログラミングにおける定石処理の例を示します。合わせて、アプリケーションノート「M16C/60 シリーズ、M16C/20 シリーズ参考プログラム集」をご利用ください。

指定したビットの状態による条件分岐

```
        BTST    0,WORK_1
        JC      LABEL1          ;指定したビットが "1" の場合 LABEL1 に分岐
          .
          .
          .
LABEL1:
        BTST    1,WORK_1
        JNC     LABEL2          ;指定したビットが "0" の場合 LABEL2 に分岐
          .
          .
          .
LABEL2:
;
```

2 命令で条件分岐される

図 4.6.1　指定したビットの状態による条件分岐プログラム例

データテーブルの検索

```
        MOV.W   #1,A0
        LDE.B   DATA_TABLE[A0],R0L ;データテーブルの 2 バイト目(34H)を R0L に格納
          .
          .
          .
DATA_TABLE:
  .BYTE  12H,34H,56H,78H        ;1 バイトデータの設定
;
```

アドレスレジスタ相対アドレッシングにより行う。
テーブルの先頭番地をベースアドレスとし、そこからの相対アドレスをアドレスレジスタ(A0,A1)に代入してデータ検索を行う。

図 4.6.2　テーブル検索プログラム例

引き数によるテーブルジャンプ

```
PARAMETER     .EQU   1
                                    飛び先番地を "LWORD" で 4 バイト設定して
                                    いるので、相対アドレス値を 4 倍する
          MOV.WPARAMETER,A0         ;引き数を A0 に設定
          SHL.W #2,A0               ;ジャンプテーブルのオフセット値算出
;
          JSRI.A JUMP_TABLE[A0]     ;テーブルジャンプ(間接サブルーチンコール)
          .                         飛び先番地の設定してあるテーブルの先頭番地をベースアド
          .                         レスとし、そこからの相対値(引数)で示されるところにある番
;                                   地へジャンプする
;=========== ROUTINE1 ==============
SUB1:
          •
          プログラム
          •
SUB1_END:
          RTS
;
;=========== ROUTINE2 ================================================
SUB2:
          .
          プログラム
          .
SUB2_END:
          RTS
;
;=========== ROUTINE3 ================================================
SUB3:
          .
          プログラム
          .
SUB3_END:
          RTS
;
;=========== ROUTINE4 ================================================
SUB4:
          .
          プログラム
          .
SUB4_END:
          RTS
;
;=========== JUMP TABLE ==========                    ============
                                    各サブルーチンの先頭アドレスをテーブル
                                    として設定しておく
JUMP_TABLE:
  .LWORD              SUB1              ;ルーチン 1
  .LWORD              SUB2              ;ルーチン 2
  .LWORD              SUB3              ;ルーチン 3
  .LWORD              SUB4              ;ルーチン 4
JUMP_TABLE_END:
```

図 4.6.3　テーブル検索プログラム例

4.7 オブジェクトファイルの生成

AS30 システムはアセンブラ(as30)、リンケージエディタ(ln30)、ロードモジュールコンバータ(lmc30)、その他のツール(lb30、abs30、xrf30)から構成される開発支援ツールです。AS30 システムを使用して、オブジェクトファイルを生成する方法を説明します。

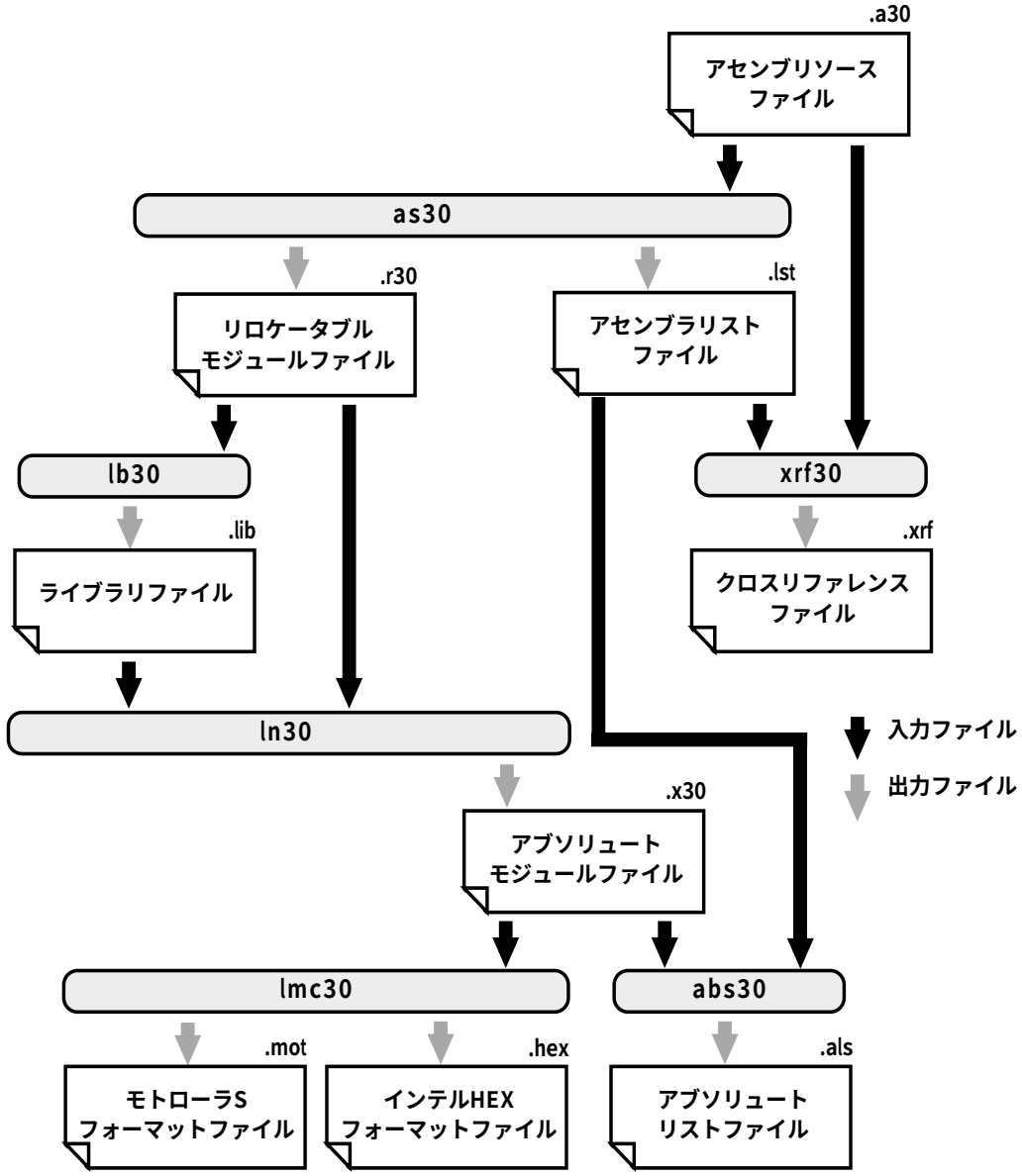


図 4.7.1　AS30 処理概要

(注)本書では AS30 システム全体を意味する場合は "AS30 システム "(大文字)と記述します。アセンブラ(as30)のみを意味する場合は "as30"(小文字)と記述します。

4.7.1 アセンブル

リロケータブルアセンブラ as30 の生成ファイルと起動方法について説明します。

as30 の生成ファイル

(1)リロケータブルモジュールファイル(*.R30)　・・・ 随時生成**
IEEE-695 に準拠したファイルで、機械語データとその再配置情報を格納したファイルです。

(2)アセンブラリストファイル(*.LST)　・・・'-L' オプション指定時**
リスト行、ロケーション情報、オブジェクトコード、行情報を含んだプリントアウト用ファイルです。

(3)アセンブラエラータグファイル(*.ATG) ・・・'-T' オプション指定時**
ソースファイルをアセンブルする際に発生したエラーメッセージを格納したファイルです。エラーが発生しなかった場合は生成されません。このファイルはタグジャンプ機能を持つエディタで利用すると、エラー修正を容易に行えます。

as30 起動方法

>as30　ファイル名 . 拡張子　〔ファイル名 . 拡張子・・・〕〔オプション〕

ファイル名は必ず 1 個以上記述してください。拡張子(.A30)は省略できます。

表 4.7.1　as30 のコマンドオプション

コマンドオプション	機能
-.	アセンブル処理のメッセージを出力しません。
-A	ニーモニックオペランドの評価をします。
-C	as30がmac30とasp30を各々起動したときのコマンドオプションを画面に表示します。
-Dシンボル名=定数	シンボル定数を設定します。
-F 展開ファイル名	指示命令..FILEの展開ファイルを固定します。
-L	-L　アセンブラリストファイルを生成します。 -LI　条件アセンブルで偽となった部分もリストに出力します。 -LM　マクロ記述の展開部分もリストに出力します。 -LIM　条件アセンブルで偽となった部分も、マクロ記述の展開部分もリストに出力します。
-M	構造化記述命令をバイト型で生成します。
-N	マクロ記述のソース行情報を、リロケータブルモジュールファイルに出力しません。
-Oディレクトリパス名	アセンブラが生成するファイルのディレクトリを指定します。 O(オウ)とディレクトリ名の間にはスペースを記述しないでくだい。 (デフォルトはカレントディレクトリ)
-P	構造化記述命令を処理します。
-S	ローカルシンボル情報をリロケータブルモジュールファイルに出力します。 -SM　システムラベル情報も出力します。
-T	タグファイルを生成します。
-V	アセンブラシステムの各プログラムのバージョンを表示します。
-Xコマンド名	エラータグファイルを生成し、コマンドを起動します。

as30 コマンド使用例

例)
各オプションはスペースで区切る

```
>as30 -L -O\work SAMPLE
```

拡張子は省略すると ".A30"

SAMPLE.A30 から SAMPLE.LST と SAMPLE.R30 を生成し、\work ディレクトリに出力する。

コマンドオプションは大文字小文字どちらでもよい

```
>as30 -sm sample
```

SAMPLE.A30 のシステムラベル情報とローカルシンボル情報を SAMPLE.R30 に出力する。

アセンブラリストファイル

アセンブラリストファイルの例を図 4.7.1 に示します。


```
* M16C FAMILY ASSEMBLER *   SOURCE LIST     Wed Mar  6 15:17:37 1996  PAGE 001

SEQ.     LOC.      OBJ.      0XMDA    . . . . *. . . . SOURCE STATEMENT. . . . 7. . . . *. . . . 8. . . . *. . . . 9 . . . . *. . . .
   1                                              ;""FILECOMMENT""*************************************************************
   2                                         ;SAMPLE PROGRAM
   3                                                  .INCLUDE m30600.inc
   4                               1                  .LIST      OFF
   5                               1                  .LIST      ON
   6                               1
   7                                         ;********************** ﾜｰｸRAM 領域確保 ******************************
   8                                                  .SECTION           WORK,DATA
   9  00400                                           .ORG               00400H
  10                                         ;
  11  00400                                 WORKRAM_TOP:
  12  00400(000001H)                        AAA:       .BLKB    1                          ;
  13  00401(000001H)                        BBB:       .BLKB    1                          ;
  14  00402(000001H)                        CCC:       .BLKB    1                          ;
  15  00403(000001H)                                   .ALIGN
  16  00404(000002H)                        DDD:       .BLKW    1                          ;
  17  00406                                 WORKRAM_END:
  18                                         ;********************** ﾋﾞｯﾄｼﾝﾎﾞﾙ定義 ************************************
  19  2,00000400h               bitsym      .BTEQU   2,AAA                          ;ﾋﾞｯﾄｼﾝﾎﾞﾙの定義
  20                                         ;********************** ｽﾀｯｸ領域確保 ************************************
  21  00000100h                             STACK_SIZE         .EQU     256
  22                                                 .SECTION           STACK,DATA
  23  01000                                          .ORG               01000H
  24  01000(000100H)                        STACK_TOP:         .BLKB    STACK_SIZE                ;ｽﾀｯｸ領域(256byte)確保
  25  00001100h                             STACK_TAIL         .EQU     STACK_TOP + STACK_SIZE
        .
        .
        .
```

インクルードファイルのネストレベルを示す

```
 * M16C FAMILY ASSEMBLER *   SOURCE LIST      Wed Mar  6 15:17:37 1996  PAGE 002

SEQ.     LOC.      OBJ.      0XMDA    ....*.... SOURCE STATEMENT.....7.....*.....8.....*.....9.....*....

 61                                  ;********************* ﾌﾟﾛｸﾞﾗﾑ領域 ***********************************
 62                                  ;======================= ｽﾀｰﾄｱｯﾌﾟﾙｰﾁﾝ ================================
 63                                          .SECTION        PROGRAM,CODE
 64  10000                                   .ORG            10000H
 65                                          .SB             00380H          ;SBﾚｼﾞｽﾀの値をｱｾﾝﾌﾞﾗに対して宣言、
 66                                          .FB             00500H          ;FBﾚｼﾞｽﾀの値をｱｾﾝﾌﾞﾗに対して宣言、
 67                                  ;
 68  10000                           START:
 69  10000  EB400011                         LDC     #STACK_TAIL,ISP         ;割り込みｽﾀｯｸﾎﾟｲﾝﾀ(ISP)値の設定
 70  10004  EB608003                         LDC     #380H,SB                ;SBﾚｼﾞｽﾀ初期値設定
 71  10008  EB700005                         LDC     #500H,FB                ;FBﾚｼﾞｽﾀ初期値設定
 72                                  ;
 73  1000C  C7030A00         S               MOV.B   #03H,PRCR               ;ﾌﾟﾛﾃｸﾄ解除
 74  10010  D97F0400          Q              MOV.W   #0007H,PM0              ;ﾌﾟﾛｾｯｻﾓｰﾄﾞﾚｼﾞｽﾀ0,1 の設定
 75                                                                          ;(RD,WRH,WRL､全ｾﾊﾟﾚｰﾄ､
 76                                                                          ;ｱﾄﾞﾚｽA19 ～ A16 出力､BCLK 出力､ｳｪｲﾄ､
 77  10014  75CF06000820                                                    ;ｸﾛｯｸﾚｼﾞｽﾀ0,1 の設定
 78                                                                          ;ｸﾛｯｸ:f(Xin),ｻﾌﾞｸﾛｯｸ
 79  1001A  B70A00           Z
 80                                  ;
 81  1001D  EB300000                         LDC     #0,FLG                  ;FLG の値設定(ｽﾀｯｸﾎﾟｲﾝﾀ:ISP を使用)
 82  10021  D9EA7D           Q*              MOV.W   #0FFFEH,PUR1            ;ﾎﾟｰﾄP44 ～ P47, ﾎﾟｰﾄP5 ～ ﾎﾟｰﾄP
 85                                  ;======================= ﾒｲﾝﾌﾟﾛｸﾞﾗﾑ ================================
 87  10024                           MAIN:
 88  10024  F50700           W               JSR     INIT                    ;初期設定ﾙｰﾁﾝ呼び出し
 89                                                                  ;(分岐範囲：-32768 ～ +32767)
 90  10027  F51400           W               JSR     DISP                    ;LED 表示ﾙｰﾁﾝ
 93                                  ;
 94  1002A                           MAIN_10:
 95  1002A  FEFF             B               JMP     MAIN_10                 ;(分岐範囲：-128 ～ +127)
 96                                  ;
          .
          .
          .
178                                 ;
179                                 .END

Information List

TOTAL ERROR(S)   00000
TOTAL WARNING(S)  00000
TOTAL LINE(S)    00179  LINES

Section List

Attr      Size       Name
DATA    0000006(00006H)  WORK
DATA    0000256(00100H)  STACK
CODE    0000083(00053H)  PROGRAM
ROMDATA  0000004(00004H)  VECT
```

Z: 命令フォーマットのゼロ形式を選択したことを示す
S: 命令フォーマットのショート形式を選択したことを示す
Q: 命令フォーマットのクイック形式を選択したことを示す

S: 分岐距離指定子 S を選択したことを示す
B: 分岐距離指定子 B を選択したことを示す
W: 分岐距離指定子 W を選択したことを示す
A: 分岐距離指定子 A を選択したことを示す

アセンブルを行った結果の全エラー数、全ワーニング数、全リスト行数を出力する

セクションのタイプ、セクションサイズ、セクション名を出力する

図 4.7.2　アセンブラリストファイルの例

アセンブルエラータグファイル

アセンブラエラータグファイルの例を図 4.7.2 に示します。

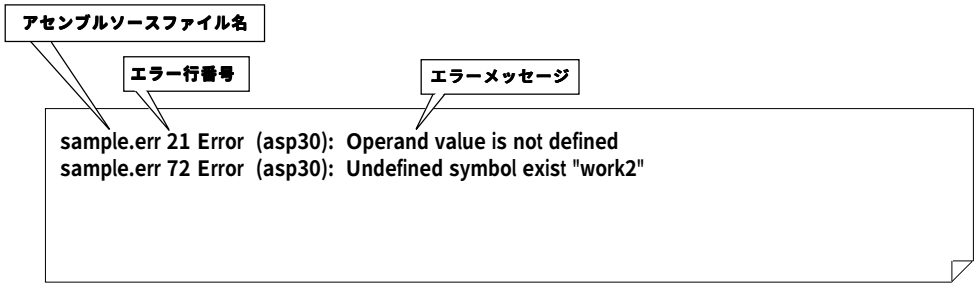


図 4.7.3　アセンブラエラータグファイルの例

4.7.2　リンク

リンケージエディタ ln30 の生成ファイルと起動方法について説明します。

ln30 の生成ファイル

(1)アブソリュートモジュールファイル(*.X30)　　・・・随時生成**

IEEE-695に準拠したファイルで、as30が出力したリロケータブルモジュールファイルを１つにまとめたファイルです。

(2)マップファイル(*.MAP)　　・・・'-M ／ -MS' オプション指定時**

リンク情報、セクションの最終配置アドレス情報、シンボル情報などを含んだファイルです。シンボル情報はオプション '-MS' を指定したときのみマップファイルに出力します。

(3)リンクエラータグファイル(*.LTG)　　・・・'-T' オプション指定時**

リロケータブルモジュールファイルをリンクする際に発生したエラーメッセージを格納したファイルです。エラーが発生しなかった場合は生成されません。このファイルはタグジャンプ機能を持つエディタで利用すると、エラー修正を容易に行えます。

ln30 起動方法

>ln30 リロケータブルファイル名 〔リロケータブルファイル名･･･〕〔オプション〕

ファイル名は必ず 1 個以上記述してください。拡張子(.R30)は省略できます。

表 4.7.2　ln30 のコマンドオプション

コマンドオプション	機能
-.	リンク処理のメッセージを出力しません。
-E　アドレス値	アブソリュートモジュールファイルの開始アドレスを設定します。オプション記号とアドレス値の間には必ずスペースを入れ、アドレス値はラベル名か16進数で記述します。
-G	ソースデバッグ情報をアブソリュートモジュールファイルに出力します。
-L　ライブラリファイル	リンク時に参照するライブラリファイルを指定します。
-LD　パス名	ライブラリファイルのディレクトリを指定します。
-M	マップファイルを生成します。マップファイル名はアブソリュートモジュールファイルの拡張子を .map に変更します。
-MS	シンボル情報を含むマップファイルを生成します。
-NOSTOP	発生したエラーを全て画面に出力します。指定しない場合はエラーを最大20個画面に出力します。
-O　アブソリュートファイル名	アブソリュートモジュールファイル名を指定します。ファイルの拡張子は省略できます。省略した場合は .x30 になります。
-ORDER	セクションの配列及び配列順序を指定します。開始アドレスを指定しない場合は、0からアドレスを配置します。
-T	エラータグファイルを出力します。
-V	バージョンを画面に表示し、そのまま終了します。
@コマンドファイル名	指定したファイルをコマンドパラメータとして起動します。@とコマンドファイル名の間にはスペースを記述しません。他のオプションとは同時に使用できません。

ln30 コマンド使用例

例)
>ln30 SAMPLE1 SAMPLE2 -O ABSSMP
ABSSMP.X30 を生成する。

拡張子 ".R30" は省略できる

コマンドオプションは大文字小文字どちらでもよい

>ln30 @cmdfile
cmdfile の内容をコマンドパラメータとして ln30 を起動する。

```
#cmdfile の記述例
SAMPLE1 SAMPLE2          # リロケータブルファイル名
SAMPLE3                  # リロケータブルファイル名
-ORDER RAM=80            #RAM セクションの開始アドレスを 80H に指定
-ORDER PROG,SUB,DATA     # セクションを配置する順序を指定
-M                       # マップファイル生成コマンドオプション
```

アドレス値は16進数で記述する。先頭の数値がアルファベットの場合は先頭に `0` を付ける。16 進数を示す `H` は付けない。

セクション名は大文字小文字を区別する

コメントの先頭には '#' を付ける

リンクエラータグファイル

リンクエラータグファイルの例を図 4.7.3 に示します。

アセンブルソースファイル名

エラー行番号

エラーメッセージ

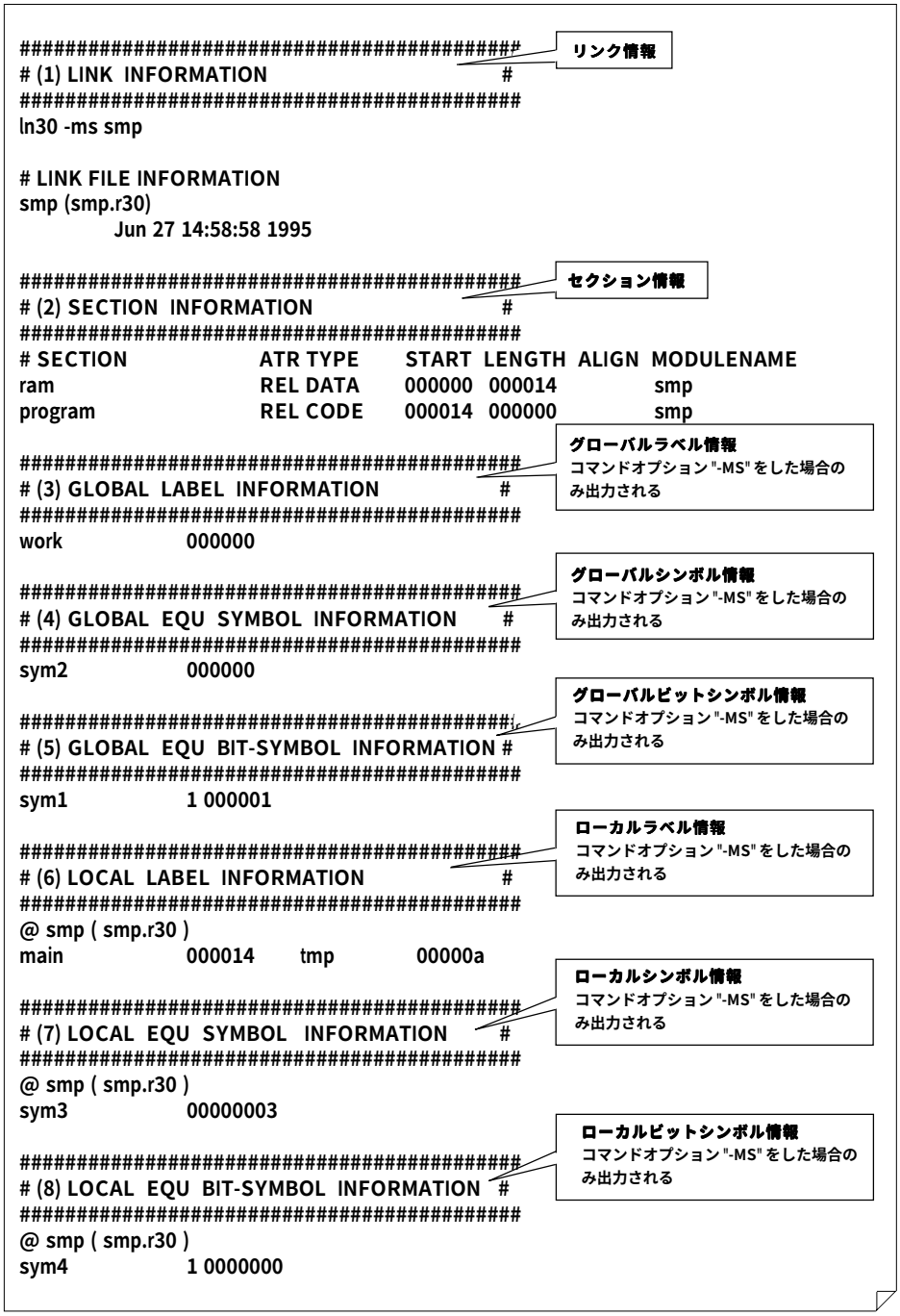
```
smp.inc 2 Warning (ln30):  smp2.r30: Absolute-section is written after the
 absolute-section 'ppp'
smp.inc 2 Error (ln30): smp2.r30: Address is overlapped in 'CODE' section 'ppp'
```

図 4.7.4　リンクエラータグファイルの例

(注)アブソリュートモジュールファイルは、IEEE-695 に準拠したフォーマットのファイルです。バイナリ形式のため、画面やプリンタに出力したり、編集したりできません。

マップファイル

マップファイルの例を図 4.7.4 に示します。




```
##############################################
# (1) LINK  INFORMATION                      #
##############################################
ln30 -ms smp

# LINK FILE INFORMATION
smp (smp.r30)
            Jun 27 14:58:58 1995

##############################################
# (2) SECTION  INFORMATION                   #
##############################################
# SECTION               ATR TYPE      START  LENGTH  ALIGN  MODULENAME
ram                     REL DATA      000000  000014          smp
program                 REL CODE      000014  000000          smp

##############################################
# (3) GLOBAL  LABEL  INFORMATION             #
##############################################
work                    000000

##############################################
# (4) GLOBAL  EQU  SYMBOL  INFORMATION       #
##############################################
sym2                    000000

##############################################
# (5) GLOBAL  EQU  BIT-SYMBOL  INFORMATION   #
##############################################
sym1                    1 000001

##############################################
# (6) LOCAL  LABEL  INFORMATION              #
##############################################
@ smp ( smp.r30 )
main                    000014        tmp           00000a

##############################################
# (7) LOCAL  EQU  SYMBOL   INFORMATION       #
##############################################
@ smp ( smp.r30 )
sym3                    00000003

##############################################
# (8) LOCAL  EQU  BIT-SYMBOL  INFORMATION    #
##############################################
@ smp ( smp.r30 )
sym4                    1 0000000
```

図 4.7.5　マップファイルの例

4.7.3　機械語ファイルの生成

ロードモジュールコンバータ lmc30 の生成ファイルと起動方法について説明します。

lmc30 の生成ファイル

(1)モトローラ S フォーマットファイル(*.MOT)　・・・通常時**
通常生成される機械語ファイルです。

(2)インテル HEX フォーマットファイル(*.HEX)　・・・`-H` オプション指定時**
'-H' オプション指定時に生成される機械語ファイルです。

lmc30 起動方法

>lmc30　[オプション]　アブソリュートモジュールファイル名

表 4.7.3　lmc30 のコマンドオプション

コマンドオプション	機能
-.	エラーメッセージを除いてすべてのメッセージを出力しない。
-E　開始アドレス	プログラムの開始アドレスを設定し、モトローラSフォーマットの機械語ファイルを生成する。-Hオプションとは同時に設定できない。
-H	拡張インテルHEXフォーマットの機械語ファイルを生成する。-Eオプションとは同時に設定できない。
-L	S2レコードで扱うことのできるデータ長を32バイトに設定する。インテルHEXフォーマットのデータ長を32バイトに設定する。
-O	lmc30が生成する機械語ファイルのファイル名を指定する。機械語ファイルはカレントディレクトリに生成する。オプションと機械語ファイル名の間には必ずスペースを入れる。機械語ファイル名の拡張子は省略できる。(モトローラSフォーマット.mot / インテルHEXフォーマット.hex)
-V	lmc30のバージョンを画面に表示しそのまま終了する。

lmc30 コマンド使用例

例)　（オプションは大文字小文字を区別しない）
>lmc30　-E 0f0000 -. DEBUG　（オプションを記述してからアブソリュートモジュールファイルを指定する）
アブソリュートモジュールファイル DEBUG.X30 から、0f0000 を開始アドレスとして機械語ファイル DEBUG.MOT を生成する。

>lmc30 -O TEST DEBUG　（拡張子 ".X30" は省略できる）
DEBUG.X30 から機械語ファイル TEST.MOT を生成する。

改訂記録	M16C/60 シリーズ、M16C/20 シリーズプログラム作成の手引き＜アセンブリ言語編＞アプリケーションノート

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2003.09.20	－	初版発行
1.01	2004.02.10	138	誤記を修正

安全設計に関するお願い

- 弊社は品質、信頼性の向上に努めておりますが、半導体製品は故障が発生したり、誤動作する場合があります。弊社の半導体製品の故障又は誤動作によって結果として、人身事故火災事故、社会的損害などを生じさせないような安全性を考慮した冗長設計、延焼対策設計、誤動作防止設計などの安全設計に十分ご留意ください。

本資料ご利用に際しての留意事項

- 本資料は、お客様が用途に応じた適切なルネサス テクノロジ製品をご購入いただくための参考資料であり、本資料中に記載の技術情報についてルネサス テクノロジが所有する知的財産権その他の権利の実施、使用を許諾するものではありません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他応用回路例の使用に起因する損害、第三者所有の権利に対する侵害に関し、ルネサス テクノロジは責任を負いません。
- 本資料に記載の製品データ、図、表、プログラム、アルゴリズムその他全ての情報は本資料発行時点のものであり、ルネサス テクノロジは、予告なしに、本資料に記載した製品または仕様を変更することがあります。ルネサス テクノロジ半導体製品のご購入に当たりましては、事前にルネサス テクノロジ、ルネサス販売または特約店へ最新の情報をご確認頂きますとともに、ルネサス テクノロジホームページ（http://www.renesas.com）などを通じて公開される情報に常にご注意ください。
- 本資料に記載した情報は、正確を期すため、慎重に制作したものですが万一本資料の記述誤りに起因する損害がお客様に生じた場合には、ルネサス テクノロジはその責任を負いません。
- 本資料に記載の製品データ、図、表に示す技術的な内容、プログラム及びアルゴリズムを流用する場合は、技術内容、プログラム、アルゴリズム単位で評価するだけでなく、システム全体で十分に評価し、お客様の責任において適用可否を判断してください。ルネサス テクノロジは、適用可否に対する責任は負いません。
- 本資料に記載された製品は、人命にかかわるような状況の下で使用される機器あるいはシステムに用いられることを目的として設計、製造されたものではありません。本資料に記載の製品を運輸、移動体用、医療用、航空宇宙用、原子力制御用、海底中継用機器あるいはシステムなど、特殊用途へのご利用をご検討の際には、ルネサス テクノロジ、ルネサス販売または特約店へご照会ください。
- 本資料の転載、複製については、文書によるルネサス テクノロジの事前の承諾が必要です。
- 本資料に関し詳細についてのお問い合わせ、その他お気付きの点がございましたらルネサス テクノロジ、ルネサス販売または特約店までご照会ください。