サンプルプログラムアプリケーションノート

■サンプルプログラム

フォルダ	実行ファイル名	項目
Sample1	Calc.psa Calc.abs	８×２キーマトリクスのキースキャン，アセンブラ関数呼び出しのサンプル printf 系関数のリンク
Sample2	Clock.psa Clock.abs	時計タイマー割込み，入力割込みのサンプル heap 領域の確保
Sample3	Disp.psa Disp.abs	文字の液晶表示，６４K以上に配置されたデータへのアクセスのサンプル
Sample4	Anime.psa Anime.abs	文字以外の液晶表示，１６Bit プログラマブルタイマ制御のサンプル

■実行プログラムの生成

●環境設定

コンパイラツールを使用するためには以下の環境設定が必要となります。
(C:¥Epson¥S1C にインストールされた場合)

```
SET PATH =C:¥Epson¥S1C88¥Bin;%PATH%
SET C88INC =C:¥Epson¥S1C88¥INCLUDE
SET C88LIB =C:¥Epson¥S1C88¥LIB
```

付属ファイルの addpath.bat に環境設定が記述されています。
ツール起動前にバッチファイルを実行してください。

●コンパイルの方法

ソースファイルの存在するフォルダ上にて DOS プロンプトを立ち上げます。
プロンプトに mk88 と入力することにより，カレントフォルダ上の makefile の内容が実行され実行プログラムが生成されます。
ソースファイルとオブジェクトファイルのタイムスタンプを比べて，ソースファイルが更新されていた場合のみ，実行ファイルの生成を行います。
ソースファイルの存在するフォルダ上に make.bat を用意してあります。
make.bat にはコンパイルツールの起動を記述してあります。
make.bat を実行することにより mk88 を起動しコンパイルができます。

■実行環境

●ハードウェア構成

本サンプルプログラムは DMT88348－DB　デモンストレーションツール上での動作が可能です。ハードウェア構成の詳細については DMT88348－DB　Demonstration Tool Manual を参照して下さい。

１．CPU

S1C88348

OSC1=32.768K　　（CLOCK サンプルで使用）

OSC3=3.68MHz　　（CALC ，DISP，ANIME サンプルで使用）

２．LCD

51×32　dot matrix

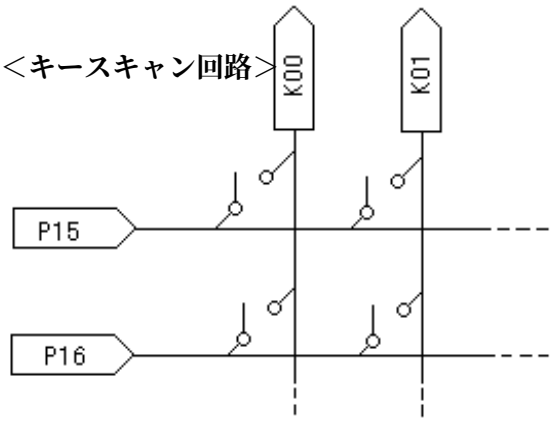
３．キーマトリクス

CALC，CLOCK［＊注］サンプルのキー配置

IN / OUT	K00	K01	K02	K03	K04	K05	K06	K07	K10	K11
P15	Esc	未使用	未使用	4	5	6	7	未使用	9	未使用
P16	1	2	3	－	=	未使用	未使用	8	未使用	0
P17	＊	／	＋	未使用	未使用	未使用	未使用	未使用	未使用	未使用
R27	A	S	D	未使用	未使用	未使用	未使用	未使用	未使用	未使用
R34	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用	未使用

各キーは図１に示すキースキャン回路を構成しています。P1x ポートが“Ｌ”出力時，キーが押されていれば対応する K ポートは“Ｈ”から“Ｌ”となります。

＜キースキャン回路＞



［＊注］

CLOCK サンプルにおいてポート R27 は K00，K01，K02（A，S，D）をキー入力割込みとして使用するため，常に“０”出力となります。

DISP，ANIME サンプルではキー入力は使用していません。

●組込みシステムシミュレータ　Sim88 でシミュレートさせる場合

サンプルプログラムが存在するフォルダの相対フォルダの simdefine フォルダ上にシミュレータに必要な定義ファイルが存在しています。

シミュレータ（Sim88）を起動後，パラメータファイルの設定画面にて ¥simdefine¥88384dmt.spj を選択して下さい。その後メニューの　File－LoadFile で　サンプルフォルダ上の *.PSA または*.ABS を選択します。

＜simdefine 上のファイル構成（Sim88 にて使用）＞

ファイル名	説明
88348DMT.CMP	コンポーネントマップファイル
88348DMT.LCD	LCD パネル定義ファイル
88348DMT.PRT	キー入力定義ファイル
88348DMT.PAR	機種情報定義ファイル
88384DMT.SPJ	シミュレータプロジェクトファイル

Sim88 の詳しい操作法はシミュレータのマニュアルを参照して下さい。

■サンプルプログラム

ＣＡＬＣ

［キーワード］

8Hz 時計タイマー割込み

８×２キーマトリクスキースキャン

アセンブラ関数呼び出し

ポーリング

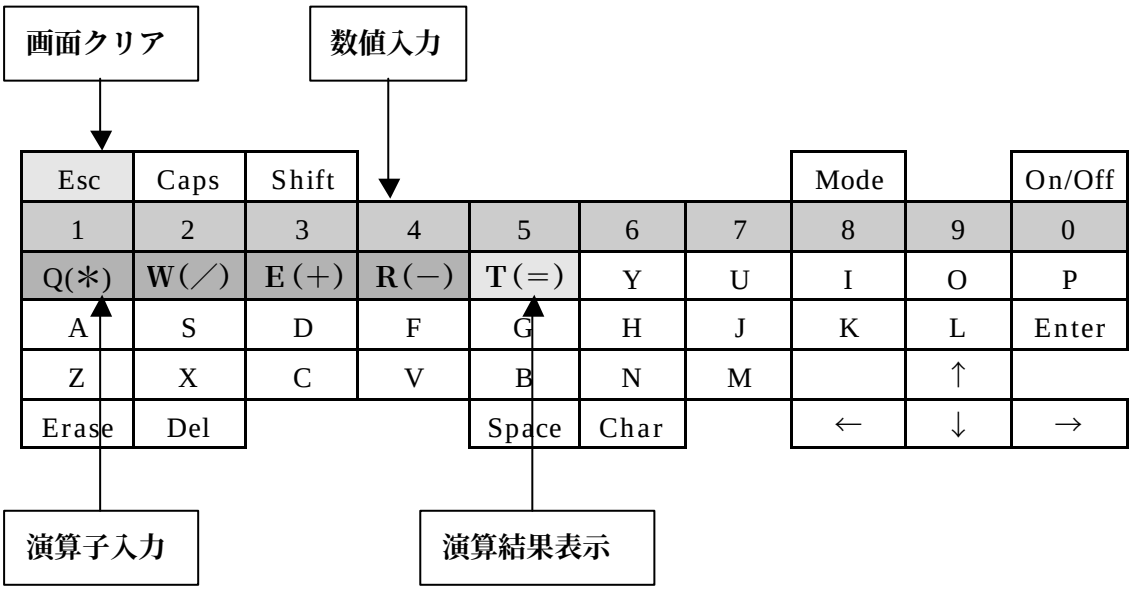
printf 系関数のリンク

［機能仕様］

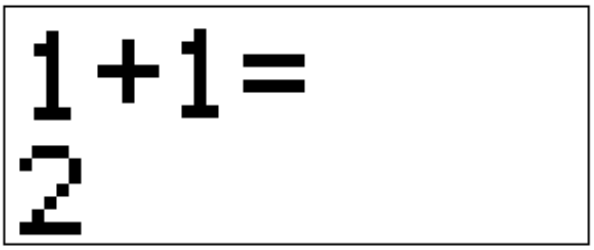
ＣＡＬＣ サンプルは、キー入力された文字の表示を行い，かつ簡単な四則演算結果を表示する簡易電卓プログラムです。

ただし入力数は 255 以下とし単項演算のみとします。

［キー操作］



［実行画面］



［プログラム仕様］

＜キースキャン＞

８×２キーマトリクスのキースキャンを使った、入力ポート（Kポート）の制御の例です。8Hz 時計タイマ割込みでキー入力をポーリングによって監視しています。

割込み内でキー入力があった場合，リングバッファにキーデータをセットします。

割込み関数終了後，メイン関数のループ内で Holt から抜けた後リングバッファ内のキーデータを液晶に表示します。また入力キーが“＝”であった場合式の演算を行い結果も液晶表示します。

またウェイト関数をアセンブラで記述し，C 言語からアセンブラ関数の呼び出し例も記述しています。

＜キースキャンの手順＞

８Hz 割込み関数内にて

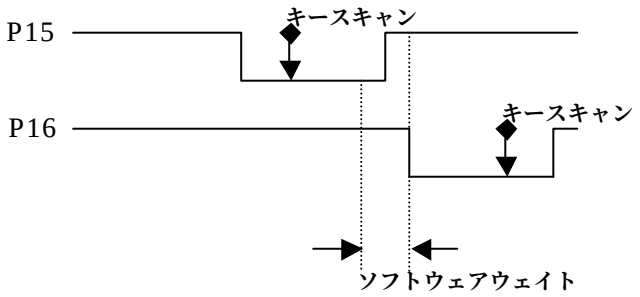
１．P15 に L 出力

２．K0x ポートデータの読み出しと保存

３．ウェイト（波形立上がり遅延吸収ディレイ）

４．P16 に L 出力

５．K0x ポートデータの読み出しと保存



＜printf 系関数のリンク＞

printf, sprintf 等の関数は文字列と引数を書式指定付で扱う単一の関数を呼び出します。この関数が_doprintf()です。この関数は書式指定子が多岐にわたるため非常におおきな間数です。書式指定子を全て利用しない場合より小さいバージョンの_doprint()関数を使用することができます。

_doprint()関数には３つのバージョンが存在します。

LARGE	完全なフォーマッタで制限がありません
MEDIUM	浮動小数点がサポートされていません
SMALL	MEDIUM とほぼ同じですが，精度指定子[.]も使用できません

使用するライブラリのリンクを指定すること選択できます。

lib/libcs

_doprnts.obj	_doprint(), small model, SMALL formatter
_doprntm.obj	_doprint(), small model, MEDIUM formatter
_doprntl.obj	_doprint(), small model, LARGE formatter
_doscans.obj	_doscan(), small model, SMALL formatter
_doscanm.obj	_doscan(), small model, MEDIUM formatter
_doscanl.obj	_doscan(), small model, LARGE formatter

lib/libcc

_doprnts.obj	_doprint(), code compact, SMALL formatter
_doprntm.obj	_doprint(), code compact, MEDIUM formatter
_doprntl.obj	_doprint(), code compact, LARGE formatter
_doscans.obj	_doscan(), code compact, SMALL formatter
_doscanm.obj	_doscan(), code compact, MEDIUM formatter
_doscanl.obj	_doscan(), code compact, LARGE formatter

lib/libcd

_doprnts.obj	_doprint(), data compact, SMALL formatter
_doprntm.obj	_doprint(), data compact, MEDIUM formatter
_doprntl.obj	_doprint(), data compact, LARGE formatter
_doscans.obj	_doscan(), data compact, SMALL formatter
_doscanm.obj	_doscan(), data compact, MEDIUM formatter
_doscanl.obj	_doscan(), data compact, LARGE formatter

lib/libcl

_doprnts.obj	_doprint(), large model, SMALL formatter
_doprntm.obj	_doprint(), large model, MEDIUM formatter
_doprntl.obj	_doprint(), large model, LARGE formatter
_doscans.obj	_doscan(), large model, SMALL formatter
_doscanm.obj	_doscan(), large model, MEDIUM formatter
_doscanl.obj	_doscan(), large model, LARGE formatter

浮動小数点を扱うには LARGE **をリンクしてください。**

CC88 –O1 -Ms ¥lib¥libcs¥_DOPRNTL.OBJ

詳しくは”makefile”**を参照して下さい。**

＜アセンブラ関数呼び出し＞

キースキャン後のウェイト関数をアセンブラで記述してあります。
アセンブラで記述した関数を C ソースから呼び出す方法を説明します。

プロトタイプ宣言　　void wait(int time);
関数コール　　wait(20*4);
関数定義

```
#pragma asm
    _wait:
        :
        RET
#pragma endasm
```

パラメータを関数に渡すときのレジスタの使用方法は固定です。
引数を渡す場合のレジスタ順序は下記のようになっています。

優先順位 / 型	高い	←			→	低い
CHAR	A	L	YP	XP	H	B
INT	BA	HL	IX	IY		
LONG	HLBA	IYIX				
Near ポインタ	IY	IX	HL	BA		
Far ポインタ	IYP	IXP	HLP			

この例では変数 time（INT 型）は，BA レジスタに値が渡されます。
仮に第２引数が LONG 型であった場合，BA が INT 型で使用しているため IYIX が使用されます。

＜使用ライブラリ関数一覧＞

関数名	ライブラリ	説明
int atoi(char*)	stdlib.h	文字列を int に変換
char* strpbrk(const char*cs, const char*ct)	string.h	文字列 ct の中にある cs の文字が最初に現れたときのポインタを返す。
char* strtok(char*s,const char*ct)	string.h	文字列 ct で区切られたトークンで文字列 s を検索
int sprintf(char*s,const char*f….)	stdio.h	文字列に一定の書式で書き込み

［コンパイルオプション］

カレントフォルダ上の makefile にコンパイルオプションが記述されています。

メモリモデル　　：small（-Ms）
出力ファイルフォーマット：Motorola S record（-srec）／IEEE-695(-ieee)
最適化オプション　　：optimization on（-O1）

［ファイルの構成］

ファイル構成は以下のようになっています。実行ファイル生成時に作成されるファイルは省略しています。

ファイル名	説明
makefile	依存関係ファイル
calc.c	サンプルプログラムソースファイル
CSTART.C	スタートアップルーチンソースファイル
ascifont.h	ASCII データ定義ヘッダファイル
S1C88348.DSC	DELFEE ファイル（ロケート時使用）
S1C88348.CPU	DELFEE ファイル（ロケート時使用）
S1C88348.MEM	DELFEE ファイル（ロケート時使用）
calc.psa	実行ファイル（Motorola S record 形式）
calc.SY	シンボル定義ファイル
calc.abs	実行ファイル（IEEE-695 形式）
make.bat	環境設定，mk88 起動バッチファイル

ＣＬＯＣＫ

［キーワード］

1Hz **時計タイマー割込み**

入力ポート割込み

Heap **領域の確保**

［機能仕様］

ＣＬＯＣＫ サンプルは、経過時間の表示を行います。

初期表示フォーマットは　“００：００：００”時間：分：秒　となっています。

またキー入力によりスタート，ストップ，リセットの動作をするストップウォッチプログラムです。プログラム起動後から,時間のカウントアップを開始します。

［キー操作］

キー	**機能**
A	**タイマースタート（タイマ動作中は無効）**
S	**タイマーストップ（タイマ停止中は無効）**
D	**表示リセット**（00:00:00）

［表示画面］

［プログラム仕様］

＜タイマー割込み＞

一秒周期で発生する時計タイマー割込み（1Hz）内で表示する秒の加算を行います。割込み関数終了後，メイン関数のループ内で Holt から復帰後液晶表示の更新を行います。また K0x ポートの入力に変化があった時に発生するキー入力割込み内でどのポートの入力があったかをフラグのビットにセットし，Holt から復帰後フラグの対応するビットがセットされていた場合時計タイマー割込みの禁止／許可，表示データのクリアを行います。

＜割込み関数の定義＞

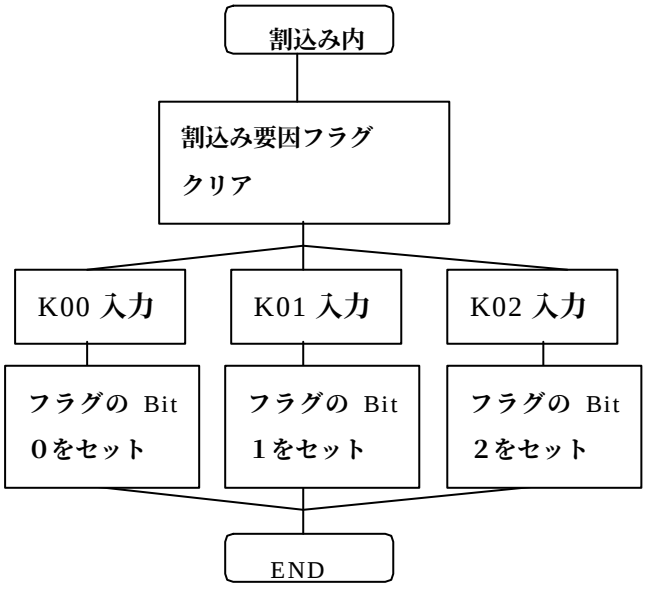
予約語_interrupt(0xXX)を使い割込み関数を定義します。

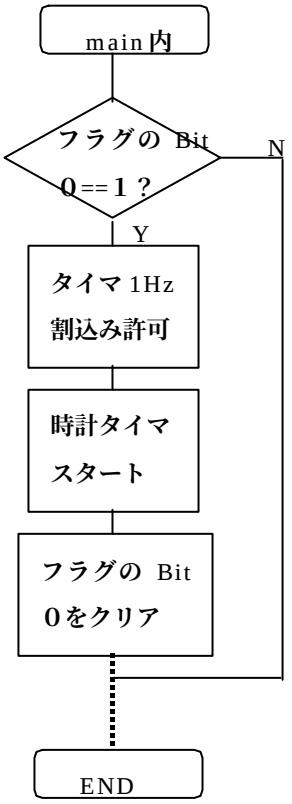
0xXX はベクタアドレスです。

void **_interrupt(0x22)** rtclock (void);：タイマー（1Hz）

void **_interrupt(0x0e)** rtkeyscan(void);：入力割込み

＜フローチャート＞





＜割込み発生タイミング＞

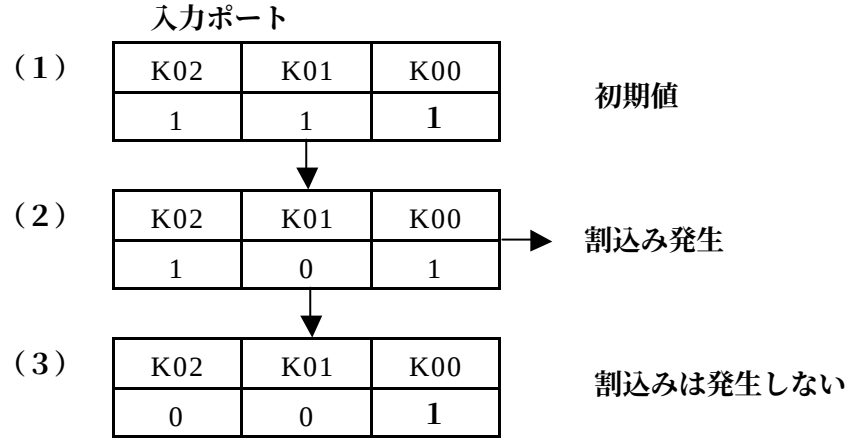
下記設定で（K00～K02）の割込みは以下のような条件で発生します。

割込み選択レジスタ

SIK02	SIK01	SIK00
1	1	1

入力比較レジスタ

KCP02	KCP01	KCP00
1	1	1



割込みは入力ポートと入力比較レジスタの内容が一致状態から不一致になることで発生します。（3）のように不一致状態から別の不一致状態に変化しても割り込みは発生しません。つまり K01 の割込み処理中に K02 のポートが変化しても再度割込みは発生しません。

＜heap 領域を使用する＞

関数 malloc を使用することによって実行時必要に応じて記憶領域の割りつけが可能です。heap が DSC ファイル上で定義されないときに malloc が使用されるとロケータがエラーを出します。特にスモールメモリモデル／コンパクトデータメモリモデルで malloc を使用する場合はロケータ記述ファイル(*.DSC)で heap の定義する場所をアドレッシングモード「データ」から「データショート」に移す必要があります。

またコンパイル時にエラーメッセージ No room for section .heap in cluster ram がでる場合は heap のサイズを指定することで回避できますので，必要なサイズを指定してください。

以下は DSC ファイルの抜粋です。

```
amode data_short {
        section selection=w;
        heap leng=100;
}
```

heap 領域に 100 バイト指定されています。

＜使用ライブラリ関数一覧＞

関数名	ライブラリ	説明
char* strcpy(char*s,const char*ct)	string.h	文字列 ct を cs にコピー。
void* malloc(size_t size)	stdlib.h	size バイトの領域さすポインタを返す。

［コンパイラオプション］

カレントフォルダ上の makefile にコンパイルオプションが記述されています。

メモリモデル　　：samll（-Ms）

出力ファイルフォーマット：Motorola S record（-srec）／IEEE-695(-ieee)

最適化オプション　　：optimization on（-O1）

［ファイルの構成］

ファイル構成は以下のようになっています。実行ファイル生成時に作成されるファイルは省略しています。

ファイル名	説明
makefile	依存関係ファイル
clkdemo.c	サンプルプログラムソースファイル
CSTART.C	スタートアップルーチンソースファイル
ascifont.h	データ定義ヘッダファイル
S1C88348.DSC	DELFEE ファイル（ロケート時使用）
S1C88348.CPU	DELFEE ファイル（ロケート時使用）
S1C88348.MEM	DELFEE ファイル（ロケート時使用）
clkdemo.psa	実行ファイル（Motorola S record 形式）
clkdemo.SY	シンボル定義ファイル
clkdemo.abs	実行ファイル（IEEE-695 形式）
make.bat	環境設定，mk88 起動バッチファイル

ＤＩＳＰ

［キーワード］

２Hz 時計タイマ

６４Ｋを超えたデータアクセス

メモリ配置の指定

ASCII 文字の液晶表示

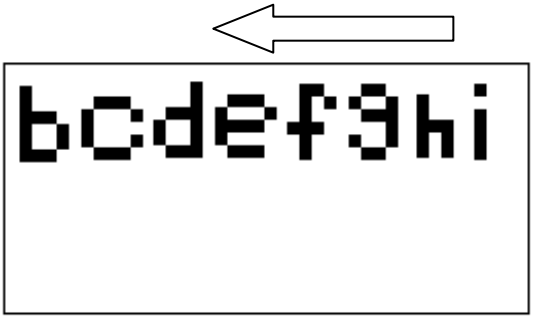
［機能仕様］

ＤＩＳＰ サンプルは、ASCII 文字（0x20～0x7d）のスクロール表示を行います。プログラム実行時からスクロールを開始しASCIIコード順に,エンドレスにスクロールし続けます。

［キー操作］

なし

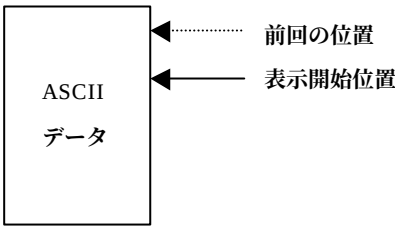
［表示画面］



［プログラム仕様］

ASCII 文字の液晶表示の例で，２Hz 時計タイマ割込み関数内で表示データの表示開始位値を加算します。割込み関数終了後，メイン関数のループ内で Holt から復帰後，表示開始位値の指し示すＡＳＣＩＩデータを液晶表示します。

また外部ＲＡＭに表示データを割り付けており，６４Ｋを超えて配置されたデータにアクセスします。



５×８ドットの文字データは ascifont.h 内で定義されており，１文字は５バイトのデータで構成されてます。

＜６４Ｋを超えたデータアクセス＞

１．メモリの配置

メモリ配置は左図のようになっており，ASCII データは 0x100000 に配置されています。

・データ定義

予約語**_far**　を使い変数を宣言することで，64K 以上の領域に配置された変数へのアクセスが可能になります。

static char **_far** AsciiFontTbl[] = {・・・};

・データの配置を指定する方法

予約語_at(0xXXXX)にて変数のアドレスを指定する方法もありますが，上記のような初期値を持った変数はアドレス指定できません。よって下記のようにします。

DSC ファイル内にてアドレスの指定を行います。

s1c88348.DSC に

section .fdata addr=0x100000;を追加。

２．システムの初期化（CSTART.C）

デフォルトでバスモードはシングルチップモードに設定されているので，設定の変更が必要です。CSTART.C ではリセットベクタおよび環境のセットアップ用として絶対コード・セクションが定義されています。リセットベクタには_START ラベルへのジャンプが含まれており，アプリケーションのデフォルト開始アドレスとして使用されています。

システムコントローラとバス制御の設定は，リセット直後の初期化ルーチンで，他の処理の前に行う必要があります。

CSTART.C の中でシステムコントローラの設定をおこなっています。

システムコントローラの設定

１．バスモード設定

２．スタックページ設定

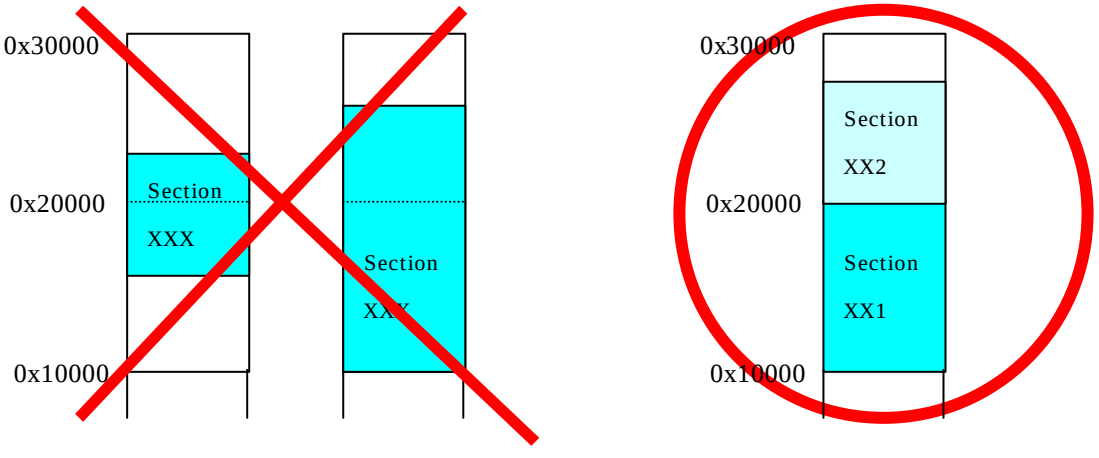
３．スタックポインタ設定

４．バス・クロック設定

＜６４K を超えるサイズのデータ＞

本コンパイラツールでは変数のサイズは 64K 以内に制限がされています。

また，サイズが 64K 以内であっても，64K の境界にまたがって配置することができません。データサイズが 64K を超えてしまうものに関してはデータを分割して配置してください。



セクション名はコンパイラがデータタイプによって決定します。

セクション名	**コメント**
.text .text_function .comm	mode s and c: コード mode d and l: コード code with _common qualifier_interrupt code
nbss fbss	cleaned_near data cleaned_far data
nbssnc fbssnc	non-cleaned_near data non-cleaned_far data
ndata fdata	initialized_near data initialized_far data
nrdata frdata	const_near data const_far data

＜セクションの分割＞

セクション名はコンパイラが決定するためセクションのサイズが, 64Kbyte **を越えてしまうことがあります。前述のように** 64Kbyte **の境界をまたいで配置することができません。その場合セクションを分割する必要があります。**

例

Test1.c **内**

```
const _far data1[60K]={…};
```

Test2.c **内**

```
const _far data2[60K]={…};
```

上記例のような定数が使われていた場合, セクション.frdata **のサイズは** 120kByte **となり，ロケーションエラーとなってしまいます。**

そこで“ -R **”オプションにてセクション名を変更することによって，セクションを分割します。**

コンパイル方法

```
c88 Test1.c -Ml -R.frdata=.frdata1
c88 Test2.c –Ml –R.frdata=.frdata2
```

［コンパイルオプション］

カレントフォルダ上の makefile にコンパイルオプションが記述されています。

メモリモデル　　：compact code（-Mc）

出力ファイルフォーマット：Motorola S record（-srec）／IEEE-695(-ieee)

最適化オプション　　：optimization on（-O1）

［ファイルの構成］

ファイル構成は以下のようになっています。実行ファイル生成時に作成されるファイルは省略しています。

ファイル名	説明
makefile	依存関係ファイル
disp.c	サンプルプログラムソースファイル
CSTART.C	スタートアップルーチンソースファイル
ascifont.h	データ定義ヘッダファイル
S1C88348.DSC	DELFEE ファイル（ロケート時使用）
S1C88348.CPU	DELFEE ファイル（ロケート時使用）
S1C88348.MEM	DELFEE ファイル（ロケート時使用）
disp.psa	実行ファイル（Motorola S record 形式）
disp.SY	シンボル定義ファイル
disp.abs	実行ファイル（IEEE-695 形式）
make.bat	環境設定，mk88 起動バッチファイル

ＡＮＩＭＥ

［キーワード］

16 ビットプログラマブルタイマ割込み

絵柄の液晶表示

C ライブラリ未使用

［機能仕様］

ＡＮＩＭＥ サンプルは、液晶表示の例で，複数の絵をパラパラ漫画のように切り替えて表示を行います。

［キー操作］

なし

［表示画面］

［プログラム仕様］

2 枚の Bitmap データを 16 ビットプログラマブルタイマ割込みを用い表示の切り替えを行います。

時計タイマ割込みでも機能的には十分実現できますが，16 ビットプログラマブルタイマ割込みを使用しています。

割込み関数内で，フラグ値の０，１を切り替えます。割込み関数終了後，メイン関数のループ内で Holt から復帰後，フラグの値によって表示するデータを選択し液晶表示を行います。また，C 標準ライブラリ関数を使用せずにアプリケーションを構築しています。

＜１６ビットリロードタイマの設定＞

割込み関数の定義

予約語**_interrupt(0xXX)** を使い割込み関数を定義します。

0xXX はベクタアドレスです。

void **_interrupt(0x06)** rtpt1 (void); ：タイマ 1

１６ビットタイマとして使用した場合，タイマ動作はタイマ 0 の設定が有効になりますが，割込みの記述はタイマ 1 のものを使います。

タイマー割込みの設定

１．プログラマブルタイマモード設定（リロードモード）

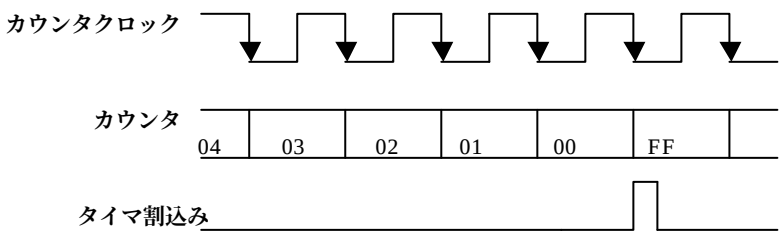
２．16Bit リロードデータ設定

３．タイマ 0 モード設定

４．割込みプライオリティ設定

５．割込みイネーブル（タイマ 1）

６．タイマ 0 スタート



＜割込み発生タイミング＞

＜C ライブラリ＞

このサンプルプログラムは C ライブラリ関数を使用していません。

C ライブラリを使用しないで実行ファイルを作成する場合以下の作業が必要になります。

1．C88 コンパイラのインストールされているフォルダ上の　Lib¥Src の中から _copytbl.asm をソースファイルのあるフォルダ上にコピーします。

2．makefile に_copytbl.asm の obj 作成の記述を追加し明示的にリンクするよう記述します。

3．CC88.EXE のオプション　–nolib を使用します。

詳しくは makefile を参照して下さい。

＜コピーテーブル＞

_copytbl()は CSTART.C の中で main()の前に呼ばれています。ROM から RAM にデータをコピーして，メモリを CLEAR 属性で初期化します。

コピーテーブルが必要な場合

RAM 領域にグローバル変数（初期値あり）が必要な場合

コピーテーブルが不必要な場合

RAM 領域にグローバル変数がない。

ROM から RAM にデータをコピーする必要がない構成のアプリケーションの場合でもデフォルトでは_copytbl()のコード自体はプログラムにロードされています。

メモリの節約をしたい場合は，CSTART.C を編集して_copytbl()の呼び出しを削除する必要があります。

［コンパイルオプション］

カレントフォルダ上の makefile にコンパイルオプションが記述されています。

メモリモデル　　：compact code（-Mc）
出力ファイルフォーマット：Motorola S record（-srec）／IEEE-695(-ieee)
最適化オプション　　：optimization on（-O1）
ライブラリ　　：ライブラリ未使用（-nolib）

［ファイルの構成］

ファイル構成は以下のようになっています。実行ファイル生成時に作成されるファイルは省略しています。

ファイル名	説明
makefile	依存関係ファイル
anime.c	サンプルプログラムソースファイル
CSTART.C	スタートアップルーチンソースファイル
anime.h	表示データ定義ヘッダファイル
_copytbl.asm	ランタイムライブラリ関数
S1C88348.DSC	DELFEE ファイル（ロケート時使用）
S1C88348.CPU	DELFEE ファイル（ロケート時使用）
S1C88348.MEM	DELFEE ファイル（ロケート時使用）
anime.psa	実行ファイル（Motorola S record 形式）
anime.SY	シンボル定義ファイル
anime.abs	実行ファイル（IEEE-695 形式）
make.bat	環境設定，mk88 起動バッチファイル

■makefile の説明

mk88.exe によってカレントフォルダの makefile に記述された内容を実行します。
makefile にはファイルの依存関係，コンパイルオプション，cc88 呼び出し等を記述します。添付の makefile の内容について説明します。
DOS プロンプトにて mk88　と入力することによりカレントフォルダ上の makefile の実行を行います。その際 mk88.EXE が存在するフォルダの相対フォルダの etc 上 mk88.mk を読み込みます。
mk88.mk には以下のマクロ定義がしてあります。

マクロ定義

```
MAKE  =     mk88
CC    =     cc88
AS    =     cc88
LD    =     cc88
CP    =     copy
RM    =     del
```

以上のマクロは makefile の中で定義しないでも使うことができます。

●条件処理

ifdef,ifndef 等が含まれる行は条件処理で使用できます。

コマンドラインから

mk88 V=1
と入力された場合 VERBOSE に-v が代入されます。

```
ifdef V
    VERBOSE= -v
endif
```

●マクロ定義部

コンパイルオプション，出力ファイル名等を定義します。
この部分を適時変更する必要があります。

```
#MACRO DEFINE#
MODEL = -Ms                 メモリモデル指定
CFILES = calc.obj ¥         リンクするオブジェクトファイル名
         cstart.obj         （複数指定可）
```

```
DSC= s1c88348.dsc                使用する DSC ファイル名
OUTPUT1= calc.psa                出力ファイル名
OUTPUT2=calc.abs
CFLAGS= -O1 $(MODEL) -c –err -g -w555
                                 コンパイルオプション
#MACRO DEFINE END#
```

バックスラッシュ“￥”にて改行をエスケープすることで複数行の入力をまたいで使用することができます。

```
CFILES= calc.obj     cstart.obj
```

●ターゲット記述部

ファイルの依存関係を記述します。

```
calc.obj : calc.c     ascifont.h                ターゲット
```

上記例では calc.c と ascifont.h のタイムスタンプが，clac.obj のそれより新しい場合のみ，次行に記述された規則を実行します。

●規則記述部

ターゲット記述に続いて規則を記述します。

＜コンパイル＞

```
calc.obj : calc.c     ascifont.h                          ターゲット
   $(CC) $(match .c $!) $(CFLAGS) $(VERBOSE)   規則
```

上記例では CC88 を呼び出して，オブジェクトファイルの生成までを行います。
コンパイルオプションはマクロ記述 CFLAGS に記述されたオプションを使用します。

```
cc88 calc.c -O1 -Ms -c –err
```

＜リンク＞

```
$(OUTPUT1) : $(CFILES) $(DSC)
     $(CC) -O1 -M $(MODEL) -o$@ $! $(SREC) $(VERBOSE)
```

OUTPUT1（calc.psa）のタイムスタンプが CFILES（_copytbl.obj calc.obj cstart.obj），DSC（s1c88f360.dsc）のタイムスタンプより古い場合，以下のコマンドを実行します。

cc88 –O1 –M –Ms calc.obj cstart.obj s1c88348.dsc –ocalc.psa –srec

ソースファイルが数本のファイルに分かれている場合，同じ数だけターゲット，規則の記述する必要があります。

●ターゲット宣言部

DOS **プロンプトにて** mk88 **と入力された場合，実行するターゲットを記述します。**

書式　all : target1　target2 ….

例

```
all: target1 target2
target1
        rule1
target2
        rule2
target3
        rule3
```

DOS **プロンプトにて** mk88 **を実行した場合** rule1,rule2 **のみが実行されます。**

特定のターゲットのみ実行したい場合

DOS **プロンプトにて**　mk88 target3 **を実行すると** rule3 **のみ実行されます。**

●関数

関数は特定の動作を実行します。構文上はマクロと同様で，次の５つの関数が使用できます。

関数名	**説明**	**例**
match	**特定の接尾辞と一致する全ての引数を生成**	$(match .obj aaa.obj bbb.c) aaa.obj **を返す**
separate	**最初の引数を区切り文字として使用し，引数を連結**	$(separate “¥n” aaa.obj bbb.c) aaa.obj bbb.c
protect	**引数を“ ”で囲います**	$(protect test) “test”
exist	**最初の引数がファイル／フォルダとして存在した場合，**2 **番目の引数を展開**	$(exit test.c cc88 test.c)
nexit	**最初の引数がファイル／フォルダとして存在しない場合，**2 **番目の引数を展開**	$(exit test.src cc88 test.c)

●特殊マクロ

規則内で省略語として使用することができます。

特殊マクロ	**説明**	calc.obj : calc.c　ascifont.h **の時返す値**
$*	**カレント・ターゲットのベース名**	calc
$<	**現在の依存関係ファイルの名前**	NULL
$@	**カレントターゲットの名前**	clac.obj
$？	**ターゲットより新しい依存関係の名前**	calc.c or ascifont.h or nothing
$！	**依存関係の名前**	calc.c　ascifont.h

例

calc.obj : calc.c　ascifont.h

$(CC) $(match .c $!) $(CFLAGS) $(VERBOSE)　　**規則**

$!は calc.c　ascifont.h **を返します。**