Application note for sample programs

■Sample programs

Folder	Execution file	Sample functions
Sample1	Calc.psa Calc.abs	8*2 key matrix key scan(polling),assembler function call, printf Formatting Routines
Sample2	Clock.psa Clock.abs	Clock timer interrupt, K port input interrupt, allocates a heap
Sample3	Disp.psa Disp.abs	Display a character on LCD read/write to data which is allocated over 64K byte
Sample4	Anime.psa Anime.abs	Display bitmap on LCD,16 bit programmable timer

■To make executable program

●environment setting

The following settings are necessary to execute the compiler tools.
(Assume the installed compiler tools are in C:¥ Epson¥S1C88 folder.)

```
SET PATH =C:¥Epson¥S1C88¥Bin;%PATH%   path for tools
SET C88INC =C:¥ Epson¥S1C88¥INCLUDE   path for include files
SET C88LIB =C:¥ Epson¥S1C88¥LIB       path for libraries
```

When a compiler tool is installed, the environment setting file (addpath.bat) is added.
Please execute addpath.bat.

●compile method

After the environment has been set, open a DOS window and change directory into the folder which contains the program source.
Input “mk88” at the DOS prompt to execute commands contained in the “makefile”.
The “makefile” compares time-stamps of source files and object files.
If source file is newer than object file, ”makefile” compiles source file.

The “make.bat” file is included with the sample program.
The “make.bat” file includes environment settings and execution compile tools.
You can make an executable object file from the source file without any environment settings by executing “make.bat”.

■Execution environment

● Target hardware

These sample programs can be executed on the Epson DMT-88348-DB demonstration tool.

For details regarding the Epson DMT-88348-DB demonstration tool, please refer to the DMT-88348-DB Demonstration Tool Manual.

1. CPU

   S1C88348

OSC1=32.768K	(used in CLOCKsample)
OSC3=3.68MHz	(used inCALC ,DISP,ANIME sample)

2. LCD

   51*32 dot matrix

3. Key matrix

   Here is key arrangement of CALC ,CLOCK [*note]

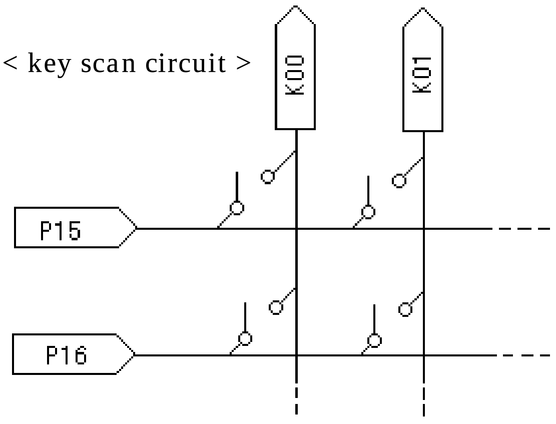
IN / OUT	K00	K01	K02	K03	K04	K05	K06	K07	K10	K11
P15	Esc	n.u.	n.u.	4	5	6	7	n.u.	9	n.u.
P16	1	2	3	-	=	n.u.	n.u.	8	n.u.	0
P17	*	/	+	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.
R27	A	S	D	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.
R34	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.

n.u.: not used

Each key is made by following the key circuit.

When “OUT” is low and the key is pressed, ”IN” will go from high to low.

< key scan circuit >



[*note]

a) In the CLOCK sample, port R27 is always low because K00, K01, K02(A, S, D) are used for the key input interrupt.

key input.

b) DISP, ANIME sample does not use key input function.

● To use Epson Embedded System Simulator Sim88

There are simulator definition files in folder “simdefine”.

When you execute simulator Sim88.exe, the simulator will ask you to set the simulator project file; set it to ¥simdefine¥88348dmt.spj.

After executing Sim88, load *.psa files in each sample folder as program data.

< simulator definition files >

File name	Note
88348DMT.CMP	Component map file
88348DMT.LCD	LCD panel definition file
88348DMT.PRT	Key input definition file
88348DMT.PAR	Debugger parameter file
88384DMT.SPJ	Simulator project file

For Sim88, refer to Epson Embedded System Simulator Operation Manual.

■Sample programs

CALC

[key words]

8Hz clock timer interrupt

8*2 key matrix key scan (polling)

calling assembler function

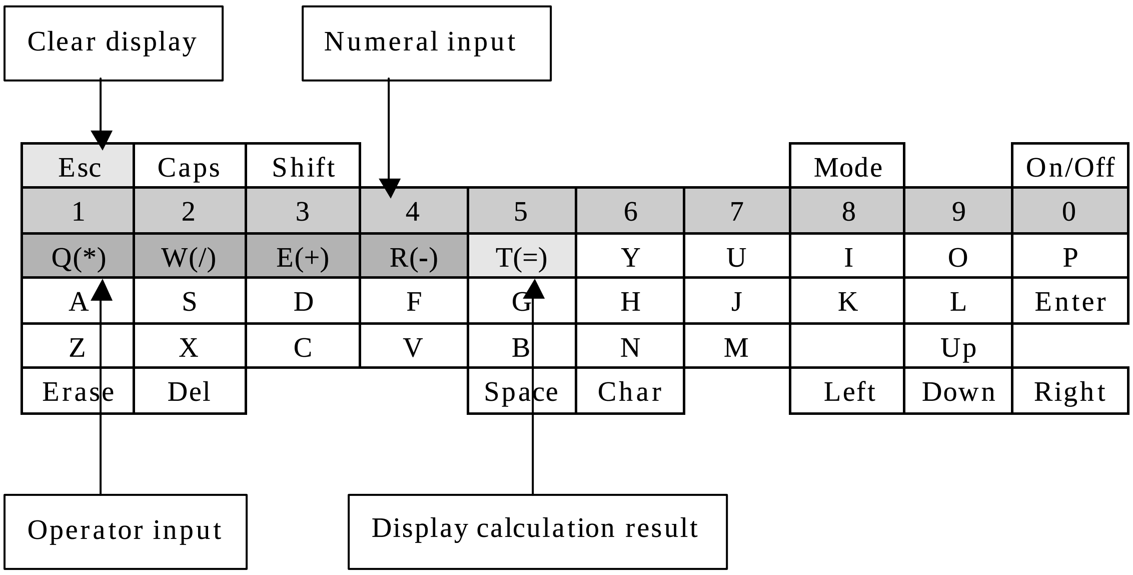
printf Formatting Routines

[function spec.]

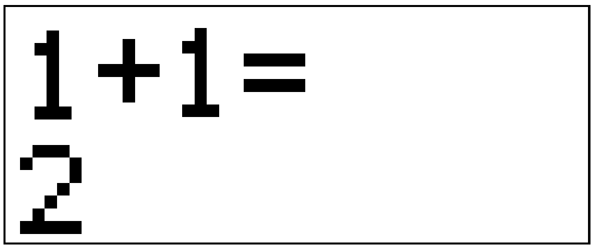
The CALC sample program displays character input by user key operation, and when “=” is input, it does simple calculation and displays the result on the LCD.

(each number should be less than256)

[key operation]



[display sample]



[program structure]

< key scan >

The following is a sample of 8*2 key matrix key scan(polling).

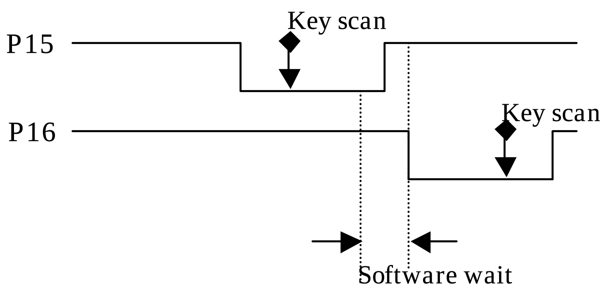
The program checks the key input port at each call to the 8Hz clock timer interrupt procedure. If the program finds any key input, this key is put into the key ring buffer. In the infinite main loop, the program finds a new key in the ring buffer. The program gets the key and display character and does the calculations when "=" key is found.

This example highlights how to call an assembler function from a C program. Note that the software wait function is an assembler function and is called from the C program.

< key scan order >

In the 8 Hz interrupt functions,

1. Write L to P15(start key scan)
2. Read K0xport data and save the data
3. Software wait (to wait P15 line level to H)
4. Write L to P16 (start key scan)
5. Read K0x port data and save the data



< printf Formatting Routines >

The functions printf(), sprintf(), ... call one single function that deals with the format string and arguments. This function is _doprint(). This is a rather big function because the number of possibilities of the format specifiers in a format string is large. If you do not use all the possibilities of the format specifiers a smaller _doprint() function can be used. Three different versions exist:

LARGE	the full formatter, no restrictions
MEDIUM	floating point printing is not supported
SMALL	Same as MEDIUM, but also the precision. specifier '.' cannot be used

You can select different formatters by linking separate objects of _doscan() and _doprint() with your application. The following objects are included:

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

Example :

```
cc88 -Ms hello.obj /c88/lib/libcs/_doprntl.obj
```

This will use the LARGE _doprint() formatter for the small model.

Please refer to “makefile”.

< how to call assembler function >

The following is an assembler function that performs software wait.

```
Prot-type declaration     void wait(int time);
Calling function          wait(20*4);
Function definition       #pragma asm
                             _wait:
                                :
                                RET
                          #pragma endasm
```

The order of register usage to pass parameter to assembler function is fixed as follows.

type \ order	1st	<<			>>	Last
CHAR	A	L	YP	XP	H	B
INT	BA	HL	IX	IY		
LONG	HLBA	IYIX				
Near pointer	IY	IX	HL	BA		
Far pointer	IYP	IXP	HLP			

In the above case, the BA register is used to pass the parameter "int time ".
(If 2nd parameter is LONG type , IXIY is used, because BA register is used to pass 1st parameter "int time".)

< used library functions >

Function name	Library	Notes
int atoi(char*)	stdlib.h	Change string to int
char* strpbrk(const char*cs, const char*ct)	string.h	Find pointer os "cs" charcter in string "ct"
char* strtok(char*s,const char*ct)	string.h	Find string "s" in tokens separated by string "ct".
int sprintf(char*s,const char*f....)	stdio.h	Write data int a string using specified format.

[compile options]

(please refer to the “makefile”.)

memory model	small (-Ms)
output file format	Motorola S record (-srec) / IEEE-695(-ieee)
optimization option	optimization (-O1)

[files in sample program folder]

(Please refer to sample program folder except in the case of derivative files to make executable file.)

File name	Explanation
makefile	“make” description file
calc.c	Sample program source
CSTART.C	Start up routine program source
ascifont.h	Ascii font data definition header file
S1C88348.DSC	DELFEE definition file
S1C88348.CPU	DELFEE definition file
S1C88348.MEM	DELFEE definition file
calc.psa	Execution file (Motorola S record)
calc.SY	Symbol definition file
calc.abs	Execution file (IEEE-695 format)
make.bat	Mk88 execution batch file

CLOCK

[key words]

1Hz clock timer interrupt

K port input interrupt

Allocates a heap

[function spec.]

CLOCK sample program display of the progress time.

Initial display format is “00:00:00” hour: minute: second.

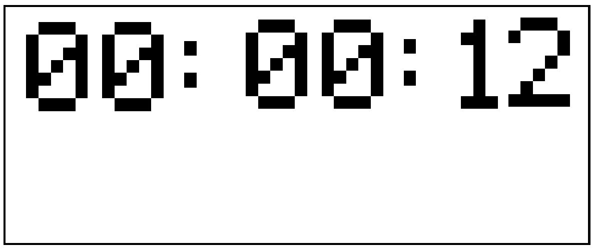
When key input, timer is start /stop/reset similarly to a Stopwatch.

After a program is started, the count of the time is started.

[key operation]

Key	Function
A	Timer start (It is invalid during timer movement.)
S	Timer stop (It is invalid during the timer stop.)
D	Display reset (00:00:00)

[display sample]



[program structure]

< timer interrupt >

1Hz clock timer interrupt procedure is called. In the infinite main loop, programs write time data to LCD memory after the interrupt procedure returns. When the K port interrupt is generated, a flag is set. This flag indicates which port received the input.

In the infinite main loop, the flag is checked.

When the flag is set, the program Starts/Stops/Resets the clock timer.

< define interrupt function >

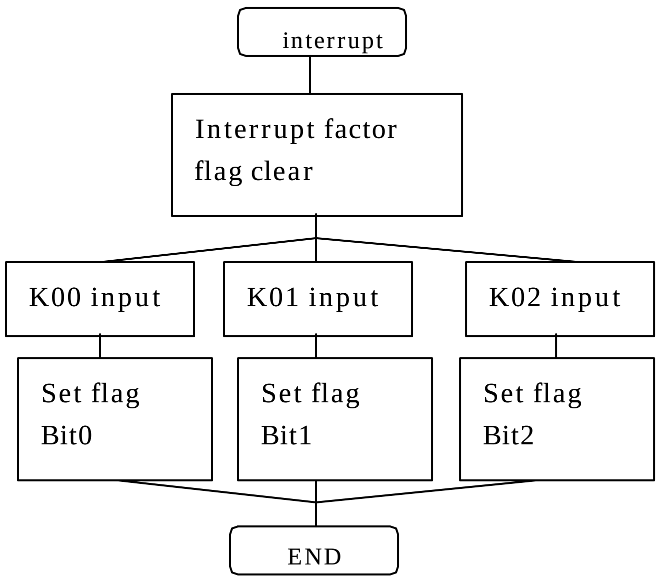
reserved word: **_interrupt(0xXX)**

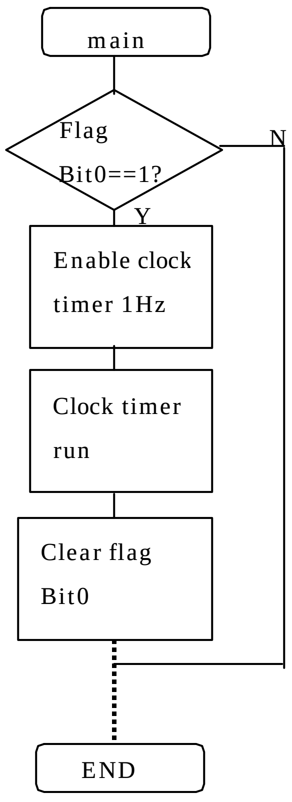
A function can be declared to serve as an interrupt service routine.

0xXX : vector address

```
void _interrupt(0x22)    rtclock (void);     :clock timer (1Hz) interrupt
void _interrupt(0x0e)    tkeyscan(void);     :Port input interrupt
```

< Flow chart >





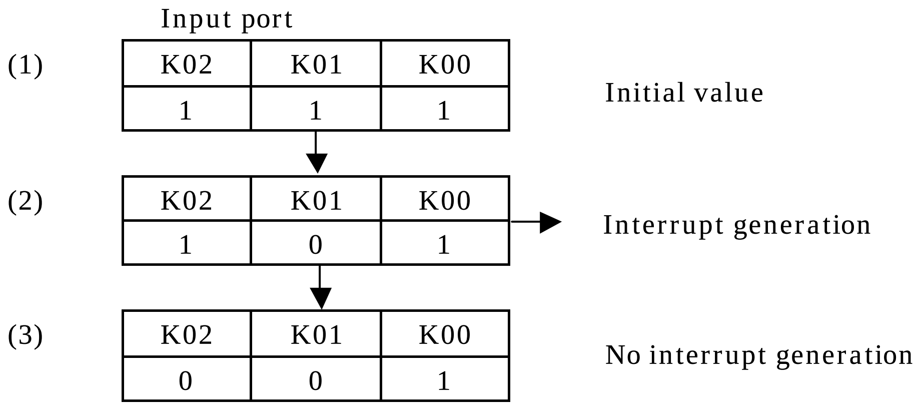
< timing (generated interrupt) >
Interrupts (K00-K02) is generated under the condition show below.

Interrupt selection register

SIK02	SIK01	SIK00
1	1	1

Input comparision register

KCP02	KCP01	KCP00
1	1	1



Interrupt will be generated when non-conformity occurs between the contents of the three bits K00-K02 and three bits input comparison register KCP00-KCP02.
Note that in case (3) of the figure, an interrupt is not generated. Consequently, in order to be able generate a second interrupt, the input comparison register KCP must be reset. When K02 port input ,interrupt not generated during K01 interrupt procedure process.

< allocates a heap >
By default no heap is allocated. When "malloc()" is used while no heap is defined, the locator gives an error. When "malloc()" is used within the small or compact data memory model, the heap from the locator description file must be moved from addressing mode 'data' to addressing mode 'data short'. Otherwise, locating the application results in locating errors.
When error message "NO room for section .heap in cluster ram" comes out, you can specify the size of the heap using the keyword length=size in the locator description file.
Here is an example part of the locator description file defining the heap size and location.

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

please refer to “makefile”.

< used library functions >

Function name	Library	Notes
char* strcpy(char*s,const char*ct)	string.h	Copies string “ct” into the string s.

[compile option]

(please refer “makefile”.)

memory model	small (-Ms)
output file format	Motorola S record (-srec) / IEEE-695(-ieee)
optimization option	optimization on(-O1)

[files in sample program folder]

(refer to the sample program folder, except for compilation derivative files)

File name	Explanation
Makefile	“make” description file
Clkdemo.c	Sample program source
CSTART.C	Start up routine program source
Ascifont.h	ASCII font data definition header file
S1C88348.DSC	DELFEE definition file
S1C88348.CPU	DELFEE definition file
S1C88348.MEM	DELFEE definition file
Clkdemo.psa	Execution file (Motorola S recordformat)
Clkdemo.SY	Symbol definition file
Clkdemo.abs	Execution file (IEEE-695 format)
make.bat	Mk88 execution batch file

DSIP

[key words]

2Hz clock timer interrupt

read/write to data which is allocated over 64k byte

Locate memory

Display character on LCD

[function spec.]

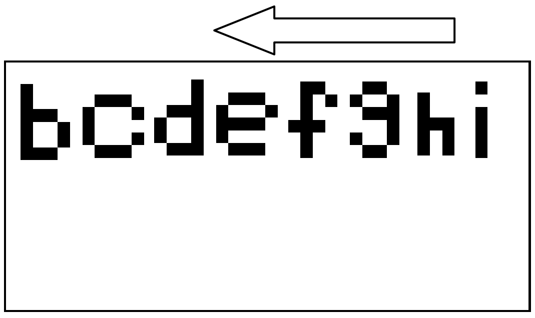
DISP scrolls the ASCII characters (0x20 - 0x7d) across the screen.

When programs start, it endlessly scrolls through the ASCII characters in order.

[key operation]

not used

[display sample]



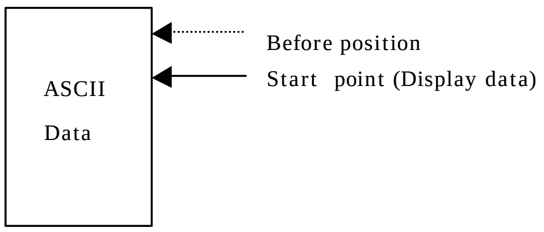
[program structure]

The following program displays characters on the LCD.

A pointer to the display start position has been added to the 2Hz clock timer interrupt procedure.

In the infinite main loop we display the characters at the start position.

We read/write data that has been allocated over 64k bytes.

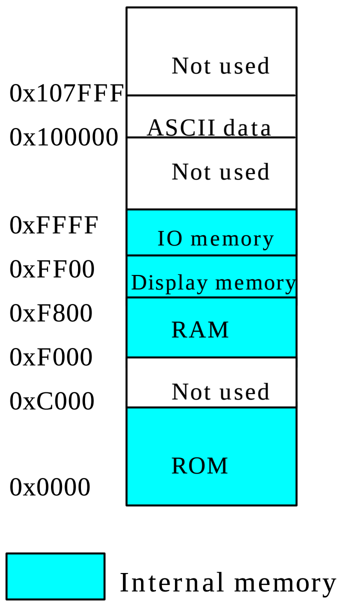


Character data(5*8 dot) are defined in asciifont.h.

It is composed of the data on five bytes per character.

< read/write to data that has been allocated over 64K byte >

1. Memory locate



Memory Address is on the left.

ASCII data is located on 0x10000.

Storage Type

Using the reserve word **_far**, anywhere in data memory, but within one 64K page

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

Locate a section at certain address

Using the reserve word _at(0xXXXX),it is possible to place certain variables at absolute addresses.

But Absolute variables cannot be initialized.

Specify the address in DSC file.

```
s1c88348.DSC
  section .fdata addr=0x100000;    Addition
```

2. Initial settings after initial reset (CSTART.C)

Bus mode is set to single chip mode by default.

In the C startup code an absolute code section is defined for setting up the reset vector in the S1C88 C environment. The reset vector contains a jump to the __START label.

This global label may not be removed, since the C compiler refers to it.

It is also used as the default start address of the application.

Settings system controller and bus mode,

You should still perform the writing operation using the initialization routine.

System controller settings

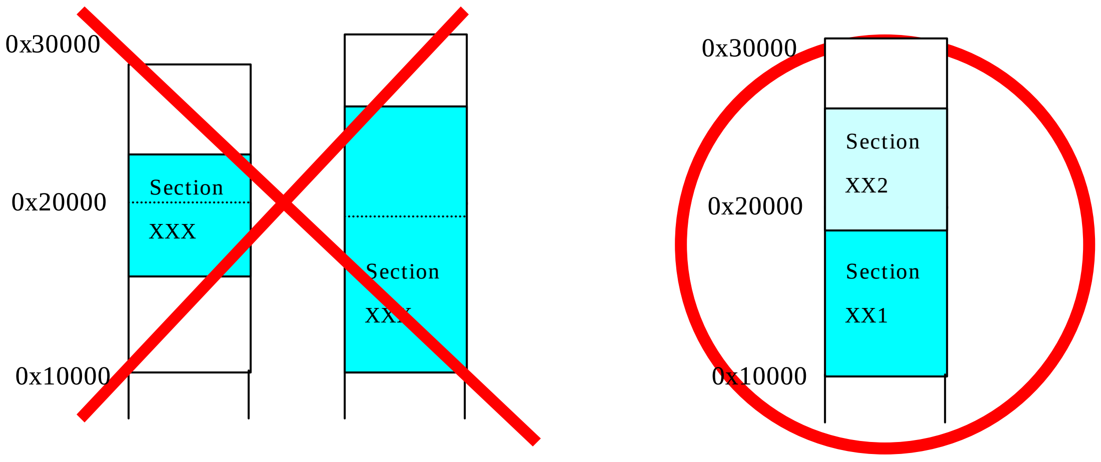
1. Bus mode setting
2. Stack page settings
3. Stack pointer settings
4. Bus clock settings

< Data which was allocated over 64K >

The size of the variable is limited to 64K for the S1C88 C environment.

When the size of the variable is within 64K, it does not cross boundaries.

If size of the variable is over 64K, you should divide the data.



For each used section the compiler generates a DEFSECT directive in the output.
The following list gives an overview of section-name used.

Section name	Comment
.text .text_function .comm	mode s and c: code mode d and l: code 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

<The division of the section >
The size of the section sometimes gets over 64KB,Because a compiler is decided as the section name.
A section can't locate the cross-boundary of 64KB.
That case section must be divided.

Example)

```
  Test1.c
        const _far data1[60K]={…};
  Test2.c
        const _far data2[60K]={…};
```

Like the above example, The size of the section .frdata becomes 120kByte.
And It becomes a location error.
A section is divided by changing a section name in the "-R" option.

Use –R Option

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

[compile option]

(please refer “makefile”.)

memory model	compact code(-Mc)
output file format	Motorola S record (-srec) / IEEE-695(-ieee)
optimization option	optimization on (-O1)

[files in sample program folder]

(Please refer to the sample program folder except for files that are derivatives of compilation.)

Filename	Explanation
Makefile	“make” description file
disp.c	Sample program source
CSTART.C	Start up routine program source
ascifont.h	ASCII font data definition header file
S1C88348.DSC	DELFEE definition file
S1C88348.CPU	DELFEE definition file
S1C88348.MEM	DELFEE definition file
disp.psa	Execution file (Motorola S recordformat)
disp.SY	Symbol definition file
disp.abs	Execution file (IEEE-695 format)
make.bat	Mk88 execution batch file

ANIME

[key words]

16bit programmable timer interrupt

Display bitmap on LCD

C library not used

[function spec.]

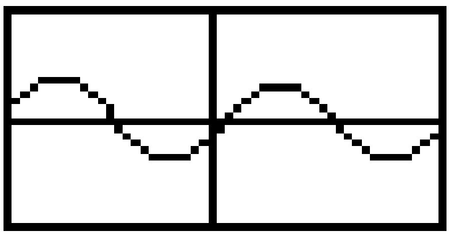
ANIME sample program displays bitmap.

More than one bitmap is changed and display.

[key operation]

not used

[display sample]



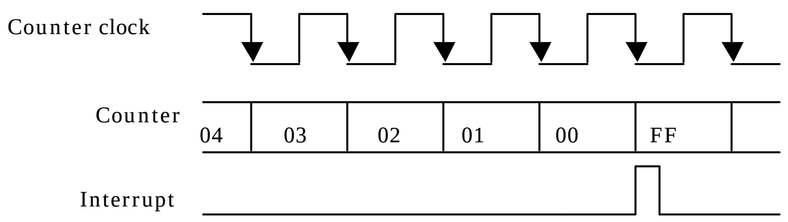
[program structure]
More than one bitmap is changed and display.
Program changes value of a flag in the 16-bit programmable timer interrupt procedure. In the infinite main loop, the program displays a bitmap on the LCD based on the value of the flag.

< 16bit reload timer settings >
Definition of the interrupt function.
reserved word: **_interrupt(0xXX)**
A function can be declared to serve as an interrupt service routine.
0xXX : vector address

void **_interrupt(0x06)** rtpt1 (void); :Timer1

When coupling programmable timers 0 and 1 for use as a 16-bit timer, the setting of timer 0 becomes valid for timer operation and setting of timer 1 becomes invalid. However, since an interrupt is generated by the underflow of timer 1, set the interrupt related routine with timer1.

settings
1. Programmable timer mode settings (reload mode)
2. 16Bit reload data settings
3. Timer 0 mode settings
4. Interrupt priority settings
5. Enable interrupt register (timer 1)
6. Run timer 0



< Timming chart >

< C Library >
This sample program does not use the C Library.

When you create an execution file without the C Library, the following steps become necessary.

1. Copy _copytbl.asm from the Lib¥Src folder relative to the folder where mk88 is located.
2. Create an object file for _copytbl.asm. and link it.
3. Use option "-nolib" for CC88.EXE.

(Please refer to "makefile".)

< copy table >
_copytbl() is called before call of main() in CSTRAT.C.
_copytbl()copies data from ROM to RAM and initializes memory with the CLEAR attribute.

_copytable is necessary in the following case:
A program has global variables that initialize RAM areas.
_copytable is unnecessary in the following case:
A program has no global variables that initialize RAM areas.

The code of _copytbl() is loaded into the program by default even in the case of the application not needing to copy data from the ROM to RAM. When you wants to save the memory, you edit CSTART.C, and you must remove the call to _copytbl().

[compile option]

(please refer “makefile”.)

memory model	compact code (-Mc)
output file format	Motorola S record (-srec) /IEEE-695 (-ieee)
optimization option	optimization on (-O1)
library	library not used (-nolib)

[files in sample program folder]

(please refer to the sample program folder except in the case of files derived from compilation.)

File name	Explanation
makefile	“make” description file
anime.c	Sample program source
CSTART.C	Start up routine program source
anime.h	Bitmap data definition header file
_copytbl.asm	Runtime library function
S1C88348.DSC	DELFEE definition file
S1C88348.CPU	DELFEE definition file
S1C88348.MEM	DELFEE definition file
anime.psa	Execution file (Motorola S record format)
anime.SY	Symbol definition file
anime.abs	Execution file (IEEE-695 format)
make.bat	Mk88 execution batch file

■ makefile

mk88 takes a file of dependencies (ie+ 'makefile') and decides what commands have to be executed to bring the files up-to-date. The following explains how to use “makefile” and mk88.
Input “mk88” on dos prompt to execute mk88 and the commands in “makefile” are executed automatically if required. Makefile reads 'mk88.mk' which is loated in the etc directory relative to the directory where mk88 is located.

The following macro-definition are in mk88.mk.

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

These macros can be use in the “makefile”.

●Conditional Processing

Lines containing ifdef, ifndef, else or endif are used for conditional processing of the makefile.

If you type the following at the dos prompt:
mk88 V=1
-v is substituted for VERBOSE.

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

●Macros

Define compile file and output file format.
You must change this part depending on your specific situation.

```
#MACRO DEFINE#
MODEL = -Ms                      memory model
CFILES = calc.obj ¥              object file name for link
          cstart.obj
DSC=  s1c88348.dsc               DSC file name
```

```
OUTPUT1= calc.psa                 outout file name
OUTPUT2=calc.abs
CFLAGS= -O1 $(MODEL) -c –err -g -w555
                                  Compile option
#MACRO DEFINE END#
```

The makefile(s) may contain a mixture of comment lines, macro definitions, include lines, and target lines. Lines may be continued across input lines by escaping the NEWLINE with a backslash (¥).

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

●**Targets**

Description file for dependencies.

```
calc.obj : calc.c   ascifont.h                Target
```

This line of the "makefile" compares source files' timestamp to calc.obj's time-stamp. If a source file is newer than object file, "makefile" executes the next line.

●**Rules**

A rule is described following the target description.

```
< compile >
calc.obj : calc.c ascifont.h                  target
   $(CC) $(match .c $!) $(CFLAGS) $(VERBOSE)   rule
```

CC88 is called and create of the object file.
A compilation option uses the option described for the macro-description CFLAGS.

```
cc88 calc.c -O1 -Ms -c –err

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

When one of the files (calc.psa,_copytbl.obj calc.obj cstart.obj,s1c88f360.dsc) is newer than calc.psa, The following command executed.

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

When a project has multiple source files, there must be a rule and target for each one.

●**Target declaration**

It is described that when you input “mk88” on dos prompt, mk88 executes the first target in the file.

```
format   all : target1 target2 ....
Example)
          all: target1 target2
          target1
                    rule1
          target2
                    rule2
          target3
                    rule3
```

When you input "mk88" on dos prompt,mk88 executed only rule1 and rule2.

If you want to execute specific target then you input "mk88 target3" on dos prompt. mk88 would only execute rule3.

●**Function**

A function not only expands but also performs a certain operation.
Functions syntactically look like macros. There are five functions.

Function	Note	Example
match	It yields all arguments which match a certain suffix.	$(match .obj aaa.obj bbb.c) Return aaa.obj
separate	It concatenates its arguments using the first argument as the separator.	$(separate “¥n” aaa.obj bbb.c) aaa.obj bbb.c
protect	Function arguments may be macros or functions themselves.	$(protect test) “test”
exist	It expands its second argument if the first argument is an existing file or directory.	$(exist test.c cc88 test.c)
nexit	It is the opposite of the exist function.	$(nexist test.src cc88 test.c)

●**Special Macros**

Several macros are useful as abbreviations within rules.

Special Macros	Note	calc.obj : calc.c ascifont.h return value
$*	The basename of the current target.	Calc
$<	The name of the current dependency file.	NULL
$@	The name of the current target.	clac.obj
$?	The names of dependents which are younger than the target.	calc.c or ascifont.h or nothing
$!	The names of all dependents.	calc.c ascifont.h

Example

```
calc.obj : calc.c   ascifont.h
$(CC) $(match .c $!) $(CFLAGS) $(VERBOSE)
```

“$!” returns calc.c ascifont.h.