BIOS Implementation for Suspend/Resume of OPL3-SA3 without APM

Ver1.20 1st August 1997
PC_Audio Dept.
YAMAHA Corporation

This is a document for BIOS in case of implementing Suspend/Resume
without APM.
To develop BIOS, there are few things to be careful.

(1) Please divide the routine in case of under APM driver or without APM.
If APM driver is exist, the device driver will save/restore all registers.
So that what the BIOS should do is, save/restore of configuration registers
of ISA PnP.

(2) If OPL3-SA3 is under the power management mode (ex. power saving mode),
please clear the power saving mode, before reading registers of YMF715.
Because under power saving mode, the registers value cannot be read correctly.

This situation can be happened like this sequence below.

```
Full on   -> Standby(OPL3-SA3 is powersaving mode)
                   -> Suspend(cannot read registers correctly)
```

To solve this problem, please clear the power management register of OPL3-SA3.
To clear the power management mode, please write "0" to Index01h.

```
Full on   -> Standby(OPL3-SA3 is powersaving mode)
               -> Standby(clear the powersaving mode of OPL3-SA3)
                   -> Suspend(cannot read registers correctly)
```

(3) As the important thing which BIOS should take care , please MASK DMA
as soon as possible, just after SMI is asserted.
If SMI is asserted when OPL3-SA3 is under DMA transferring, there is a possibility
that the condition of DMA(bank, counter address etc) is not saved correctly.
To avoid this situation, please mask DMA just after SMI is asserted.
Maybe BIOS need to go through some sequence before saving Audio registers,
but at least, masking DMA should be put the first procedure of the BIOS
after SMI is asserted.
And also, put unmaskDMA to the last sequence of Resume routine.
Putting unmuteDMA just before exiting SMI is better.

Note:

If the system has OPL4ML(YMF704) or OPL4ML2(YMF721) with YMF715, and
its design is /SEL4&7, in this case, Suspend/Resume of OPL4ML(2) is supported.
If BIOS supports Suspend/Resume of OPL4ML(2) without APM, please go thorough
its procedure. Sample code is attached by assembler code as “gmpproc2.asm”.

OPL4ML2 supports power saving mode. If the system is using OPL4ML2, and the
system needs to support power saving mode of OPL4ML2 , please go thorough its procedure.
Sample code is attached by assembler code as “gmpproc2.asm”.

If your system has implemented OPL4ML(2) with /SEL5, Suspend/Resume cannot be done perfectly, but when resuming please send midi message of “note off”.　Because there is a possibility that inside the OPL4ML(2) is unstable after resuming.　Sample code is attached as “all_note_off.asm”

The procedure to Suspend

(1) If OPL3-SA3 is under power management mode,
    clear the power management register(Index01h).
(2) Mask DMA
(3) Save configuration register value
(4) Save current Index value.
(5) Save Mixer value, and mask master volume.
(6) Save OPL3 registers.
(7) Save Sound Blaster Pro registers.
(8) Save MPU registers.
(9) save WSS registers.
(10) save control registers.

The procedure to Resume

(1) Restore configuration registers.
(2) Wait for WSS becomes active.
(3) Restore Mixer registers
(4) Restore OPL3 registers
(5) Restore Sound Blaster Pro registers
(6) Restore MPU registers
(7) Restore WSS registers
(8) Restore Control registers
(9) Unmask DMA
(10) Unmute Master Volume
(11) Restore current Index value.

There are several attached files

(1) *susres.h*

Header file of sample C code (suspend.c, resume.c)
Please take a look of page 6.

(2) *suspend.c*

Sample C code for Suspend without APM.
Please take a look of page 9.

(3) *resume.c*

Sample C code for Resume without APM
Please take a look of page 21.

(4) *gmpproc2.asm*

Sample Assembler code for Suspend/Resume for which system has OPL4ML or OPL4ML2 with /SEL4,7.
Please take a look of page 34.

(5) all_note_off.asm

Sample Assembler code for Suspend/Resume for which system has OPL4ML or OPL4ML2 with /SEL5.
Please take a look of page 44.

(6) Assembler code for Suspend/Resume

These assembler files are developed based on "suspend.c" and "resume.c". The sequence itself is same as C language source code.
Please take a look of page 47.

This page is intentionally blank.

SUSRES.H

```
//		Suspend Resume Header for OPL3-SA3 (YMF715)
//
//		Copyright (C) 1997, YAMAHA Corporation.
//		All rights reserved.
//
//		Compiler:	Borland C++ 3.1J
//		History:			Version 1.0  March 18th, 1997
//						Newly complied.
//							Version 1.10		May 8th, 1997
//						Add SB mixer register
//							Version 1.20		Aug 1st, 1997
//						Correct Scandata array size

#define BYTE		unsigned char
#define	WORD	unsigned int
#define	DWORD	unsigned long

unsigned char		yamahaKey[32] =
{
 0xb1,0xd8,0x6c,0x36,0x9b,0x4d,0xa6,0xd3,
 0x69,0xb4,0x5a,0xad,0xd6,0xeb,0x75,0xba,
 0xdd,0xee,0xf7,0x7b,0x3d,0x9e,0xcf,0x67,
 0x33,0x19,0x8c,0x46,0xa3,0x51,0xa8,0x54
};
unsigned char		initiationKey[32] =
{
 0x6a,0xb5,0xda,0xed,0xf6,0xfb,0x7d,0xbe,
 0xdf,0x6f,0x37,0x1b,0x0d,0x86,0xc3,0x61,
 0xb0,0x58,0x2c,0x16,0x8b,0x45,0xa2,0xd1,
 0xe8,0x74,0x3a,0x9d,0xce,0xe7,0x73,0x39
};

struct	configurationRegister{
 BYTE	active0;
 WORD	sbBase;
 WORD	wssBase;
 WORD	adlibBase;
 WORD	mpuBase;
 WORD	ctrlBase;
 BYTE	irqA;
 BYTE	irqB;
 BYTE	dmaA;
 BYTE	dmaB;
 BYTE	active1;
 WORD	joyBase;
};

struct	mixerRegister{
 BYTE	masterL;
 BYTE	masterR;
 BYTE	inputL;
 BYTE	inputR;
 BYTE	aux1L;
 BYTE        aux1R;
 BYTE        aux2L;
 BYTE        aux2R;
 BYTE        waveL;
 BYTE        waveR;
 BYTE        lineL;
 BYTE        lineR;
 BYTE        mono;
 BYTE        mic;
 BYTE        wide;
 BYTE        bass;
 BYTE        tre;
};

// modefied May 7th
struct       sbRegister{
 BYTE        voice;
 BYTE        mic;
 BYTE        source;
 BYTE        sw;
 BYTE        master;
 BYTE        midi;
 BYTE        cd;
 BYTE        line;
};

struct       wssRegister{
 BYTE        index;
 BYTE        mode;
 BYTE        interface;
 BYTE        status;
 BYTE        playFormat;
 BYTE        wssiControl;
 BYTE        playBaseUpper;
 BYTE        playBaseLower;
 BYTE        playCurrentUpper;
 BYTE        playCurrentLower;
 BYTE        recFormat;
 BYTE        recBaseUpper;
 BYTE        recBaseLower;
 BYTE        recCurrentUpper;
 BYTE        recCurrentLower;
 BYTE        dacConfig;
 BYTE        timerUpper;
 BYTE        timerLower;
 BYTE        iStatus;
 BYTE        iControl;
};

struct       powerManagement{
 BYTE        clock;
 BYTE        down;
 BYTE        save;
 BYTE        aSave;
 BYTE       part1;
 BYTE       part2;
};

struct      controlRegister{
 BYTE       index;
 BYTE       system;
 BYTE       irqChannel;
 BYTE       dmaChannel;
 BYTE       misc;
};

struct      opl3Registers{
 BYTE       mode;
 BYTE       nts;
 BYTE       timer1;
 BYTE       timer2;
 BYTE       tCtrl;
 BYTE       mult[2][18];
 BYTE       tl[2][18];
 BYTE       ad[2][18];
 BYTE       sr[2][18];
 BYTE       fnum[2][9];
 BYTE       block[2][9];
 BYTE       fb[2][9];
 BYTE       ws[2][18];
 BYTE       rhythm;
 BYTE       connect;
};

struct      configurationRegister cfg;
struct      mixerRegister mix;
struct      sbRegister sb;
struct      wssRegister wss;
struct      powerManagement pm;
struct      controlRegister ctrl;
struct      opl3Registers opl;

BYTE        dmaAState;
BYTE        dmaBState;
BYTE        mpuMode;

// Modified Aug 1st
BYTE        scanData[28];
BYTE        resourceData[512];

BYTE        opl3Slot[18] = {
 0x00,0x01,0x02,0x03,0x04,0x05,
 0x08,0x09,0x0a,0x0b,0x0c,0x0d,
 0x10,0x11,0x12,0x13,0x14,0x15
};
```

SUSPEND.C

```
//------------------------------------------------------------------------
//          Suspend Routine for OPL3-SA3 (YMF715)
//
//          Copyright (C) 1997, YAMAHA Corporation.
//          All rights reserved.
//
//          Compiler:  Borland C++ 3.1J
//          History:    Version 1.0           March 18th, 1997
//                                   Newly complied.
//                                 Version 1.10          May 8th, 1997
//                                   Add SB mixer register, Correction MPU
//                                 Version 1.11          May 9th, 1997
//                                   correction MPU
//                                 Version 1.12     May 9th, 1997
//                                   correction OPL
//                                 Version 1.20          May 21st, 1997
//                                   correction DMA mask
//                                   changes priority of DMA mask
//                                 Version 1.21          Aug 1st, 1997
//                                   Correct write size of scanData in main
//
//------------------------------------------------------------------------
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>
#include "susres.h"

//------------------------------------------------------------------------
//          Wait Loop
//          1-time means about 2 us
//------------------------------------------------------------------------
void        wait(DWORD time)
{
 int        i;

 for(i=0;i<time;i++)
            inp(0x388);            // Any I/O port is OK
}

//------------------------------------------------------------------------
//          Write to PnP Configuration Register
//------------------------------------------------------------------------
void        cfgWR(BYTE index, BYTE data)
{
 outp(0x279, index);
 outp(0xa79, data);
}

//------------------------------------------------------------------------
//          Read from PnP Configuration Register
//------------------------------------------------------------------------
```

```
BYTE        cfgRD(BYTE index)
{
 outp(0x279,index);
 return     (BYTE)inp(0x203);
}

//---------------------------------------------------------------------
//          Write to WSS register
//---------------------------------------------------------------------
void        wssWR(BYTE index, BYTE data)
{
 outp(cfg.wssBase+4, index);
 outp(cfg.wssBase+5, data);
}

//---------------------------------------------------------------------
//          Read from WSS register
//---------------------------------------------------------------------
BYTE        wssRD(BYTE index)
{
 outp(cfg.wssBase+4, index);
 return (BYTE)inp(cfg.wssBase+5);
}

//---------------------------------------------------------------------
//          Write to Control Register
//---------------------------------------------------------------------
void        ctrlWR(BYTE index, BYTE data)
{
 outp(cfg.ctrlBase, index);
 outp(cfg.ctrlBase+1, data);
}

//---------------------------------------------------------------------
//          Read from Control Register
//---------------------------------------------------------------------
BYTE        ctrlRD(BYTE index)
{
 outp(cfg.ctrlBase, index);
 return (BYTE)inp(cfg.ctrlBase+1);
}

//---------------------------------------------------------------------
//          Write to OPL3 Register
//
//          Argument 'bank' means OPL3 register array (0 or 1)
//---------------------------------------------------------------------
void        oplWR(BYTE bank, BYTE index, BYTE data)
{
 if(bank==0)
            outp(cfg.adlibBase, index);
 else if(bank==1)
            outp(cfg.adlibBase+2, index);
 wait(4);
 outp(cfg.adlibBase+1, data);
 wait(4);
}

//------------------------------------------------------------------------
//              Read from OPL3 Register
//
//              Argument 'bank' means OPL3 register array (0 or 1)
//------------------------------------------------------------------------
BYTE        oplRD(BYTE bank, BYTE index)
{
 if(bank==0)
                outp(cfg.adlibBase, index);
 else if(bank==1)
                outp(cfg.adlibBase+2, index);
 wait(4);

 return (BYTE)inp(cfg.adlibBase+1);

}

//------------------------------------------------------------------------
//              modified May 8th
//              Read from SB Mixer Register
//------------------------------------------------------------------------
BYTE        sbRD(BYTE index)
{
 outp(cfg.sbBase+4, index);
 return (BYTE)inp(cfg.sbBase+5);
}


//------------------------------------------------------------------------
//              DMA channel Mask Routine
//
//              If DMA channels used by OPL3-SA3 are not mask, mask these DMA
//              channel
//------------------------------------------------------------------------
void        maskDMA(void)
{
 // modified on May 21st, 1997
 // dmaAState = (!inp(0x0f)>>cfg.dmaA)&0x01;            // 1: dma used   0: not used
 // dmaBState = (!inp(0x0f)>>cfg.dmaB)&0x01;            // 1: dma used   0: not used
 dmaAState = ( (~  inp(0x0f))>>cfg.dmaA )&0x01;          // 1: dma used   0: not used
 dmaBState = ( (~  inp(0x0f))>>cfg.dmaB )&0x01;          // 1: dma used   0: not used

 if(dmaAState)
                outp(0x0a,(0x04|cfg.dmaA));             // dmaA channel mask
 if(dmaBState)
                outp(0x0a,(0x04|cfg.dmaB));             // dmaB channel mask

 // Added on May 21st, 1997
 // Wait until DMA stops completely.
 // time = max 250 usec   @ Fs=4kHz, 8bit, mono
 wait(125);
}

//--------------------------------------------------------------------------
//            Read Current Index register value
//
//            If PC system switches to suspend routine just after index register
//            is set, PC system need to know current index register value for
//            resume routine. Otherwise, after resume, data is set to wrong
//            register.
//--------------------------------------------------------------------------
void          currentIndex(void)
{
 wss.index = (BYTE)inp(cfg.wssBase+4);
 ctrl.index = (BYTE)inp(cfg.ctrlBase);
}

//--------------------------------------------------------------------------
//            Wake-up OPL3-SA3 Routine
//
//            If you switched OPL3-SA3 from stand-by state to suspend (powered
//            off), you need this routine. This routine forces OPL3-SA3 to switch
//            normal state (Full ON) from any state.
//--------------------------------------------------------------------------
void          wakeup(void)
{
 pm.clock = ctrlRD(0x01)&0x01;                        // 1:Clock stop    0:Clock stop
 pm.down = (ctrlRD(0x01)>>1)&0x01;
 pm.save = (ctrlRD(0x01)>>2)&0x01;
 pm.aSave = (ctrlRD(0x01)>>5)&0x01;
 pm.part1 = ctrlRD(0x12);
 pm.part2 = ctrlRD(0x13);

 if(pm.clock)
 {
              ctrlWR(0x01, ctrlRD(0x01)&(!0x01));
              wait(50000);              // 100 ms wait
 }

 if(pm.down)
 {
              ctrlWR(0x01, ctrlRD(0x01)&(!0x01));
              wait(10);    // 20 us wait
 }

 if(pm.save)
 {
              ctrlWR(0x01, ctrlRD(0x01)&(!0x01));
              wait(10);    // 20 us ms wait
 }

 if(pm.part1)
 {
```

```
                ctrlWR(0x12,0x00);
                wait(10);    // 20 us wait
 }
}

//----------------------------------------------------------------------
//              Send YAMAHA key to switch OPL3-SA3 to Wait for Key state
//----------------------------------------------------------------------
void            sendYamahaKey(void)
{
 int            i;

 //--- reset pnp configuration ----
 cfgWR(0x02,0x02);
 wait(2000);

 outp(0x279,0x00);
 outp(0x279,0x00);

 for(i=0;i<32;i++)
                outp(0x279,yamahaKey[i]);
}

//----------------------------------------------------------------------
//              Read Configuration Routine
//
//              Read current configuration of OPL3-SA3 using YAMAHA Key manner.
//              Select the Readport to any I/O space that no device is present.
//----------------------------------------------------------------------
int             suspendConfig(void)
{
 int            nodetect;
 int            i;

 nodetect = 1;
 sendYamahaKey();

 // Wake up OPL3-SAx series device
 cfgWR(0x03,0x81);

 wait(10);

 // force to set 0x203 as Read_Port
 cfgWR(0x00,0x80);

 wait(10);
 if(cfgRD(0x06)!=0x81)
                return nodetect;

 //-----
 // If you need,
 // read back the resource data from internal SRAM
 //-----
 for(i=0;i<512;i++)
```

```
{
        resourceData[i] = cfgRD(0x04);
        wait(2);
}

//--------
// read back configuration register value
// LDN = 0
cfgWR(0x07,0x00);
cfg.sbBase = (WORD)cfgRD(0x61) + (cfgRD(0x60)<<8);
cfg.wssBase = (WORD)cfgRD(0x63) + (cfgRD(0x62)<<8);
cfg.adlibBase = (WORD)cfgRD(0x65) + (cfgRD(0x64)<<8);
cfg.mpuBase = (WORD)cfgRD(0x67) + (cfgRD(0x66)<<8);
cfg.ctrlBase = (WORD)cfgRD(0x69) + (cfgRD(0x68)<<8);
cfg.irqA = cfgRD(0x70);
cfg.irqB = cfgRD(0x72);
cfg.dmaA = cfgRD(0x74);
cfg.dmaB = cfgRD(0x75);
cfg.active0 = cfgRD(0x30);
if(cfg.active0!=0x00)
        nodetect = 0;

// LDN = 1
cfgWR(0x07,0x01);
cfg.joyBase = (cfgRD(0x60)<<8)|cfgRD(0x61);
cfg.active1 = cfgRD(0x30);

cfgWR(0x02,0x02);
return nodetect;
}

//------------------------------------------------------------------------
//        Suspend Mixer Routine
//
//        Read current Mixer setting of OPL3-SA3. At first, mute the Master
//        Volume.
//------------------------------------------------------------------------
void      suspendMixer(void)
{

 mix.masterL = ctrlRD(0x07);
 mix.masterR = ctrlRD(0x08);

 //--- mute master volume ----
 ctrlWR(0x07, (0x80|mix.masterL));
 ctrlWR(0x08, (0x80|mix.masterR));

 wss.mode = (wssRD(0x0c)>>6)&0x01;

 mix.inputL = wssRD(0x00);
 mix.inputR = wssRD(0x01);
 mix.aux1L = wssRD(0x02);
 mix.aux1R = wssRD(0x03);
 mix.aux2L = wssRD(0x04);
 mix.aux2R = wssRD(0x05);
 mix.waveL = wssRD(0x06);
 mix.waveR = wssRD(0x07);

 if(wss.mode)
 {
               mix.lineL = wssRD(0x12);
               mix.lineR = wssRD(0x13);
               mix.mono = wssRD(0x1a);
 }

 mix.mic = ctrlRD(0x09);
 mix.wide = ctrlRD(0x14);
 mix.bass = ctrlRD(0x15);
 mix.tre = ctrlRD(0x16);

}

//------------------------------------------------------------------------
//             modified May 8th
//             Suspend SBpro Mixer Routine
//
//             Read current SBpro Mixer setting of OPL3-SA3.
//------------------------------------------------------------------------
void           suspendsbMixer(void)

{
 sb.voice = sbRD(0x04);
 sb.mic = sbRD(0x0a);
 sb.source = sbRD(0x0c);
 sb.sw = sbRD(0x0e);
 sb.master = sbRD(0x22);
 sb.midi = sbRD(0x26);
 sb.cd = sbRD(0x28);
 sb.line = sbRD(0x2e);
}

//------------------------------------------------------------------------
//             Suspend SB routine
//
//             All internal state of SB portion can be read by using scan register.
//             The size of scanned data is 218bit.
//------------------------------------------------------------------------
void           suspendSB(void)
{
 int i,j,t;

 t = 2;

 ctrlWR(0x10,0x01);                        // set SBPDR
 while (1) {
               if ((ctrlRD(0x10) & 0x80) == 0x80) break;
 }
 ctrlWR(0x10,0x0D);                          // ss=1 sm=1 se=0 sbpdr=1
 for (i = 0;i < 27;++i)
 {
             for (j = 0;j < 8;++j)            // generate 8 clocks
             {
                         ctrlWR(0x10,0x0F);   // ss=1 sm=1 se=1
                         ctrlWR(0x10,0x0D);   // ss=1 sm=1 se=0
                         wait(t);
             }
             scanData[i] = ctrlRD(0x11);      // read byte in shift register
 }

 for (j = 0;j < 2;++j)                        // generate the last clocks
 {
             ctrlWR(0x10,0x0F);               // ss=1 sm=1 se=1
             ctrlWR(0x10,0x0D);               // ss=1 sm=1 se=0
             wait(t);
 }
 scanData[i] = (ctrlRD(0x11) & 0x03) << 6;
 ctrlWR(0x10,0x01);                           // ss=0 sm=0 se=0

}

//-----------------------------------------------------------------------
//           Suspend MPU routine
//
//           Only the operation mode need to be stored before power off.
//           To investigate the operation mode is writing the reset command.
//           If MPU401 is not in UART mode, MPU401 returns acknowledge (FEh)
//           from command register. Otherwise, MPU401 doesn't return acknowledge.
//-----------------------------------------------------------------------
void         suspendMPU(void)
{

 outp(cfg.mpuBase+1,0xff);                    // Modified: May 2nd, 1997
//           if((!inp(cfg.mpuBase+1))&0x80)              // Modified: May 2nd, 1997
// modified May 9th
 if((inp(cfg.mpuBase+1)&0x80)==0x00 )                    // Modified: May 2nd, 1997
             mpuMode = 1;                     // UART mode
 else
             mpuMode = 0;                     // not UART mode

}

//-----------------------------------------------------------------------
//           Suspend WSS routine
//
//           At first, check whether WSS still do the playback or the capture.
//           Then if do, stop the playback and the capture.
//           WSS portion has the counter in order to count the number of
//           transferred data. The current value of this is read from control
//           register.
//-----------------------------------------------------------------------
void         suspendWSS(void)
```

```
{

 wss.interface = wssRD(0x09);

//		if(wss.interface&0x03)
//				wssWR(0x09,0x0);	// Stop playback & capture

 wss.status = (BYTE)inp(cfg.wssBase+6);

 wss.playFormat = wssRD(0x08);
 wss.iControl = wssRD(0x0a);
 wss.playCurrentLower = ctrlRD(0x0b);
 wss.playCurrentUpper = ctrlRD(0x0c);
 //wait(10);
 wss.playBaseLower = wssRD(0x0f);
 wss.playBaseUpper = wssRD(0x0e);

 if(wss.mode)
 {
		wss.recFormat = wssRD(0x1c);
		wss.recBaseUpper = wssRD(0x1e);
		wss.recBaseLower = wssRD(0x1f);
		wss.recCurrentUpper = ctrlRD(0x0e);
		wss.recCurrentLower = ctrlRD(0x0d);
		wss.dacConfig = wssRD(0x10);
		wss.timerUpper = wssRD(0x15);
		wss.timerLower = wssRD(0x14);
		wss.iStatus = wssRD(0x18);
 }

}

//------------------------------------------------------------------------
//		Suspend OPL3 registers
//
//		The current Address value can not be read from OPL3 portion.
//-----------------------------------------------------------------------
void		suspendOpl3(void)
{
 int		i;

 opl.mode = oplRD(1,0x05);

 opl.nts = oplRD(0,0x08);
 opl.timer1 = oplRD(0,0x02);
 opl.timer2 = oplRD(0,0x03);
 opl.tCtrl = oplRD(0,0x04);

 for(i=0;i<18;i++)
		opl.mult[0][i] = oplRD(0,opl3Slot[i]+0x20);
 for(i=0;i<18;i++)
		opl.tl[0][i] = oplRD(0,opl3Slot[i]+0x40);
 for(i=0;i<18;i++)
		opl.ad[0][i] = oplRD(0,opl3Slot[i]+0x60);
 for(i=0;i<18;i++)
          opl.sr[0][i] = oplRD(0,opl3Slot[i]+0x80);
 for(i=0;i<9;i++)
          opl.fnum[0][i] = oplRD(0,i+0xa0);
 for(i=0;i<9;i++)
          opl.block[0][i] = oplRD(0,i+0xb0);
 for(i=0;i<9;i++)
          opl.fb[0][i] = oplRD(0,i+0xc0);
 for(i=0;i<18;i++)
          opl.ws[0][i] = oplRD(0,opl3Slot[i]+0xe0);

 opl.rhythm = oplRD(0,0xbd);

 if( (opl.mode&0x01) == 0x01 )
 {
          // modified May 9th Ver1.12
          //opl.connect = oplRD(1,0x06);
          opl.connect = oplRD(1,0x04);

          // Modified argument of oplRD: May 2nd, 1997
          for(i=0;i<18;i++)
                   opl.mult[1][i] = oplRD(1,opl3Slot[i]+0x20);
          for(i=0;i<18;i++)
                   opl.tl[1][i] = oplRD(1,opl3Slot[i]+0x40);
          for(i=0;i<18;i++)
                   opl.ad[1][i] = oplRD(1,opl3Slot[i]+0x60);
          for(i=0;i<18;i++)
                   opl.sr[1][i] = oplRD(1,opl3Slot[i]+0x80);
          for(i=0;i<9;i++)
                   opl.fnum[1][i] = oplRD(1,i+0xa0);
          for(i=0;i<9;i++)
                   opl.block[1][i] = oplRD(1,i+0xb0);
          for(i=0;i<9;i++)
                   opl.fb[1][i] = oplRD(1,i+0xc0);
          for(i=0;i<18;i++)
                   opl.ws[1][i] = oplRD(1,opl3Slot[i]+0xe0);
 }

}

//------------------------------------------------------------------------
//        Suspend Control Register
//------------------------------------------------------------------------
void      suspendCtrl(void)
{

 ctrl.system = ctrlRD(0x02);
 ctrl.irqChannel = ctrlRD(0x03);
 ctrl.dmaChannel = ctrlRD(0x06);
 ctrl.misc = ctrlRD(0x0a);
}

//------------------------------------------------------------------------
//        Main Routine
```

```
//---------------------------------------------------------------------------
void            main(void)
{
 FILE           *fp;
 WORD           data;

 //--------------------------
 // If OPL3-SA3 stays in Power Down state, wake up OPL3-SA3 before
 // execution of this suspend routine. In this case, you need to
 // know the Base address of Control register. Otherwise, you can't
 // wake up OPL3-SA3.
 // We recommend to store the Base address of Control register in
 // SMRAM area before entering stand-by.
 //
 // If(oplStandby==ENABLE)
 //             wakeup();
 //
 // oplStandby is the variable that is present in global area.


 // Modified on May 22, 1997
 // Priority was changed. DMA should be masked at first.

 maskDMA();
                printf("Both DMA channels are masked\n");


 // This routine is used for only test.
 //
 if(suspendConfig())
 {
                printf("OPL3-SA3 was not found or was not activated.\n");
                return;
 }
 else
                printf("The configuration is stored.\n");
 //

 currentIndex();
//                              printf("The current index is stored.\n");
 suspendMixer();
//                              printf("The mixer setting is stored.\n");
//              modified May 8th
 suspendsbMixer();
//                              printf("The SBpro mixer setting is stored.\n");
 suspendOpl3();
//                              printf("The OPL register is stored.\n");
 suspendSB();
//                              printf("The SB state is stored.\n");
 suspendMPU();
//                              printf("The MPU401 state is stored.\n");
 suspendWSS();
//                              printf("The WSS register is stored.\n");
 suspendCtrl();
//                              printf("The Control register is stored\n");


 if( (fp=fopen("susres.dat","wb")) == NULL )
 {
                printf("Can't open susres.dat\n");
                return;
 }

 fwrite(resourceData,512,1,fp);
 fwrite(&cfg,18,1,fp);
 fwrite(&mix,18,1,fp);
//              modified May 8th
 fwrite(&sb,8,1,fp);
 fwrite(&wss,21,1,fp);
 fwrite(&opl,241,1,fp);
 fwrite(&pm,6,1,fp);
 fwrite(&ctrl,5,1,fp);
//              modified Aug 1st
fwrite(scanData,28,1,fp);
 putc(dmaAState,fp);
 putc(dmaBState,fp);
 putc(mpuMode,fp);
 fclose(fp);

}
```

RESUME.C

```
//-----------------------------------------------------------------------
//          Resume Routine for OPL3-SA3 (YMF715)
//
//          Copyright (C) 1997, YAMAHA Corporation.
//          All rights reserved.
//
//          Compiler:  Borland C++ 3.1J
//          History:   Ver 1.00    March 18th, 1997
//                                   Newly complied.
//                                 Ver 1.01 May 2nd, 1997
//                                   Correction of WSS resume routine
//                                   Correction of OPL3 Array number
//                                 Ver 1.10 May 8th, 1997
//                                   Add SB mixer register
//                                 Ver 1.11 May 9th, 1997
//                                   Correction MPU, WSS resume routine
//                                 Ver 1.12 May 9th 1997
//                                   Correction OPL
//                                 Ver 1.20 May 22, 1997
//                                   Correction WSS resume routine
//                                 Ver 1.21 Aug 1st, 1997
//                                   Correction read size of scanData in Main
//
//
//-----------------------------------------------------------------------
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dos.h>
#include "susres.h"

//-----------------------------------------------------------------------
//          Wait Loop
//          1-time means about 2 us
//-----------------------------------------------------------------------
void        wait(DWORD time)
{
 int        i;

 for(i=0;i<time;i++)
            inp(0x388);             // Any I/O port is OK
}

//-----------------------------------------------------------------------
//          Write to PnP Configuration Register
//-----------------------------------------------------------------------
void        cfgWR(BYTE index, BYTE data)
{
 outp(0x279, index);
 outp(0xa79, data);
}
```

```
//--------------------------------------------------------------------
//          Read from PnP Configuration Register
//--------------------------------------------------------------------
BYTE        cfgRD(BYTE index)
{
 outp(0x279,index);
 return      (BYTE)inp(0x203);
}

//--------------------------------------------------------------------
//          Write to WSS register
//--------------------------------------------------------------------
void        wssWR(BYTE index, BYTE data)
{
 outp(cfg.wssBase+4, index);
 outp(cfg.wssBase+5, data);
}

//--------------------------------------------------------------------
//          Read from WSS register
//--------------------------------------------------------------------
BYTE        wssRD(BYTE index)
{
 outp(cfg.wssBase+4, index);
 return (BYTE)inp(cfg.wssBase+5);
}

//--------------------------------------------------------------------
//          Write to Control Register
//--------------------------------------------------------------------
void        ctrlWR(BYTE index, BYTE data)
{
 outp(cfg.ctrlBase, index);
 outp(cfg.ctrlBase+1, data);
}

//--------------------------------------------------------------------
//          Read from Control Register
//--------------------------------------------------------------------
BYTE        ctrlRD(BYTE index)
{
 outp(cfg.ctrlBase, index);
 return (BYTE)inp(cfg.ctrlBase+1);
}

//--------------------------------------------------------------------
//          Write to OPL3 Register
//
//          bank means OPL3 register array (0 or 1)
//--------------------------------------------------------------------
void        oplWR(BYTE bank, BYTE index, BYTE data)
{
 if(bank==0)
            outp(cfg.adlibBase, index);
 else if(bank==1)
                outp(cfg.adlibBase+2, index);
 wait(4);

 outp(cfg.adlibBase+1, data);
 wait(4);
}

//---------------------------------------------------------------------
//              Read from OPL3 Register
//---------------------------------------------------------------------
BYTE            oplRD(BYTE bank, BYTE index)
{
 if(bank==0)
                outp(cfg.adlibBase, index);
 else if(bank==1)
                outp(cfg.adlibBase+2, index);
 wait(4);

 return (BYTE)inp(cfg.adlibBase+1);

}

//---------------------------------------------------------------------
//              modified May 8th
//              Write to SBpro Mixer Register
//---------------------------------------------------------------------
void            sbWR(BYTE index, BYTE data)
{
 outp(cfg.sbBase+4, index);
 outp(cfg.sbBase+5, data);
}

//---------------------------------------------------------------------
//              DMA channel Mask Routine
//
//              Current DMA channels used by OPL3-SA3 are needed as the arguments.
//---------------------------------------------------------------------
void            unmaskDMA(int dmaA, int dmaB)
{

 if(dmaAState)
                outp(0x0a,dmaA);                // dmaA channel unmask
 if(dmaBState)
                outp(0x0a,dmaB);                // dmaB channel unmask
}

//---------------------------------------------------------------------
//              Read Current Index register value
//
//              If PC system switches to suspend routine just after index register
//              is set, PC system need to know current index register value for
//              resume routine. Otherwise, after resume, data is set to wrong
//              register.
```

```
//------------------------------------------------------------------------
void		currentIndex(void)
{
 wss.index = (BYTE)inp(cfg.wssBase+4);
 ctrl.index = (BYTE)inp(cfg.ctrlBase);
}

//------------------------------------------------------------------------
//		Send YAMAHA key to switch OPL3-SA3 to Wait for Key state
//------------------------------------------------------------------------
void		sendYamahaKey(void)
{
 int		i;

 //--- reset pnp configuration ----
 cfgWR(0x02,0x02);
 wait(2000);

 outp(0x279,0x00);
 outp(0x279,0x00);

 for(i=0;i<32;i++)
		outp(0x279,yamahaKey[i]);
}

//------------------------------------------------------------------------
//		Write Configuration Routine
//
//		Write previous configuration of OPL3-SA3 using YAMAHA key manner.
//		Select the Readport to any I/O space that no device appears.
//------------------------------------------------------------------------
int		resumeConfig(void)
{
 int		nodetect;
 int		i;
 DWORD	deviceID;

 nodetect = 1;
 sendYamahaKey();

 // Wake up OPL3-SAx series device
 cfgWR(0x03,0x81);

 wait(10);

 // force to set 0x203 as Read_Port
 cfgWR(0x00,0x80);

 wait(10);
 if(cfgRD(0x06)!=0x81)
		return nodetect;

 //-----
 // If you need,
// write back the resource data to internal SRAM
//-----
cfgWR(0x21,0x01);
wait(10);
for(i=0;i<512;i++)
{
        cfgWR(0x20,resourceData[i]);
        wait(2);
}
cfgWR(0x21,0x00);

//--------
// write back configuration register value
// LDN = 0
cfgWR(0x07,0x00);
cfgWR(0x60,(BYTE)(cfg.sbBase>>8));
cfgWR(0x61,(BYTE)(cfg.sbBase));
cfgWR(0x62,(BYTE)(cfg.wssBase>>8));
cfgWR(0x63,(BYTE)(cfg.wssBase));
cfgWR(0x64,(BYTE)(cfg.adlibBase>>8));
cfgWR(0x65,(BYTE)(cfg.adlibBase));
cfgWR(0x66,(BYTE)(cfg.mpuBase>>8));
cfgWR(0x67,(BYTE)(cfg.mpuBase));
cfgWR(0x68,(BYTE)(cfg.ctrlBase>>8));
cfgWR(0x69,(BYTE)(cfg.ctrlBase));
cfgWR(0x70,cfg.irqA);
cfgWR(0x72,cfg.irqB);
cfgWR(0x74,cfg.dmaA);
cfgWR(0x75,cfg.dmaB);
cfgWR(0x30,cfg.active0);
if(cfg.active0!=0x00)
        nodetect = 0;

// LDN = 1
cfgWR(0x07,0x01);
cfgWR(0x60,(BYTE)(cfg.joyBase>>8));
cfgWR(0x61,(BYTE)(cfg.joyBase));
cfgWR(0x30,cfg.active1);

cfgWR(0x02,0x02);
wait(0x2000);

return nodetect;

}

//---------------------------------------------------------------------------
//      Resume Mixer Routine
//
//      Write previous Mixer setting of OPL3-SA3. At first, mute the Master
//      Volume.
//---------------------------------------------------------------------------
void    resumeMixer(void)
{
wssWR(0x0c,(wss.mode<<6));

//--- mute master volume ----
ctrlWR(0x07, (0x80|mix.masterL));
ctrlWR(0x08, (0x80|mix.masterR));

wssWR(0x00,mix.inputL);
wssWR(0x01,mix.inputR);
wssWR(0x02,mix.aux1L);
wssWR(0x03,mix.aux1R);
wssWR(0x04,mix.aux2L);
wssWR(0x05,mix.aux2R);
wssWR(0x06,mix.waveL);
wssWR(0x07,mix.waveR);

if(wss.mode)
{
        wssWR(0x12,mix.lineL);
        wssWR(0x13,mix.lineR);
        wssWR(0x1a,mix.mono);
}

ctrlWR(0x09,mix.mic);
ctrlWR(0x14,mix.wide);
ctrlWR(0x15,mix.bass);
ctrlWR(0x16,mix.tre);

}

//-------------------------------------------------------------------------
//              modified May 8th
//              Resume SBpro Mixer Routine
//
//              Read current SBpro Mixer setting of OPL3-SA3.
//-------------------------------------------------------------------------
void            resumeSBmixer(void)

{
sbWR(0x04,sb.voice);
sbWR(0x0a,sb.mic);
sbWR(0x0c,sb.source);
sbWR(0x0e,sb.sw);
sbWR(0x22,sb.master);
sbWR(0x26,sb.midi);
sbWR(0x28,sb.cd);
sbWR(0x2e,sb.line);
}

//-------------------------------------------------------------------------
//              Resume OPL3 Routine
//-------------------------------------------------------------------------
void            resumeOPL(void)
{
```

```
int             i;

oplWR(1,0x05,opl.mode);

oplWR(0,0x08,opl.nts);
oplWR(0,0x02,opl.timer1);
oplWR(0,0x03,opl.timer2);

for(i=0;i<18;i++)
                oplWR(0,opl3Slot[i]+0x20,opl.mult[0][i]);
for(i=0;i<18;i++)
                oplWR(0,opl3Slot[i]+0x40,opl.tl[0][i]);
for(i=0;i<18;i++)
                oplWR(0,opl3Slot[i]+0x60,opl.ad[0][i]);
for(i=0;i<18;i++)
                oplWR(0,opl3Slot[i]+0x80,opl.sr[0][i]);
for(i=0;i<9;i++)
                oplWR(0,i+0xa0,opl.fnum[0][i]);
for(i=0;i<9;i++)
                oplWR(0,i+0xb0,(opl.block[0][i]&0xdf));
for(i=0;i<9;i++)
                oplWR(0,i+0xc0,opl.fb[0][i]);
for(i=0;i<18;i++)
                oplWR(0,opl3Slot[i]+0xe0,opl.ws[0][i]);

oplWR(0,0xbd,opl.rhythm);

if( (opl.mode&0x01) == 0x01 )
{

                // modified May 9th Ver1.12
                //oplWR(1,0x06,opl.connect);
                oplWR(1,0x04,opl.connect);

                for(i=0;i<18;i++)
                                oplWR(1,opl3Slot[i]+0x20,opl.mult[1][i]);       // Modified: May 2nd, 1997
                for(i=0;i<18;i++)
                                oplWR(1,opl3Slot[i]+0x40,opl.tl[1][i]);                         // Modified: May 2nd,
1997
                for(i=0;i<18;i++)
                                oplWR(1,opl3Slot[i]+0x60,opl.ad[1][i]);                         // Modified: May 2nd,
1997
                for(i=0;i<18;i++)
                                oplWR(1,opl3Slot[i]+0x80,opl.sr[1][i]);                         // Modified: May 2nd,
1997
                for(i=0;i<9;i++)
                                oplWR(1,i+0xa0,opl.fnum[1][i]);                                 // Modified: May 2nd,
1997
                for(i=0;i<9;i++)
                                oplWR(1,i+0xb0,(opl.block[1][i]&0xdf));                         // Modified: May 2nd,
1997
                for(i=0;i<9;i++)
                                oplWR(1,i+0xc0,opl.fb[1][i]);                                   // Modified: May 2nd,
1997
```

```
                for(i=0;i<18;i++)
                                oplWR(1,opl3Slot[i]+0xe0,opl.ws[1][i]);                         // Modified: May 2nd,
1997
 }

 oplWR(0,0x04,opl.tCtrl);

}


//-------------------------------------------------------------------------
//              Resume SB Routine
//-------------------------------------------------------------------------
void            resumeSB(void)
{
 int i,j,t;

 t = 2;

 ctrlWR(0x10,0x09);                     // ss=1 sm=0 se=0 sbpdr=1
 for(i = 0;i < 27;++i)
 {
                ctrlWR(0x11, scanData[i]);
                for(j = 0;j < 8;++j)       // generate 8 clocks
                {
                                ctrlWR(0x10,0x0B);     // ss=1 sm=0 se=1 sbpdr=1
                                ctrlWR(0x10,0x09);     // ss=1 sm=0 se=0 sbpdr=1
                                wait(t);
                }
 }
 ctrlWR(0x11, scanData[i]);
 for (j = 0;j < 2;++j)                  // generate the last clocks
 {
                ctrlWR(0x10,0x0B);     // ss=1 sm=0 se=1 sbpdr=1
                ctrlWR(0x10,0x09);     // ss=1 sm=0 se=0 sbpdr=1
                wait(t);
 }

 ctrlWR(0x10,0x00);                      // ss=0 sm=0 se=0 sbpdr=0
}

//-------------------------------------------------------------------------
//              Resume MPU routine
//-------------------------------------------------------------------------
void            resumeMPU(void)
{
 int            loop;

 if(mpuMode)
 {
                loop = 10;
                outp(cfg.mpuBase+1,0x3f);
                while(loop--)
                {
```

```
//                                      if( (!inp(cfg.mpuBase+1))&0x80 )
// modified May 9th
                           if( (inp(cfg.mpuBase+1)&0x80)==0x00 )
                           {
                                      if(inp(cfg.mpuBase)==0xfe)
                                                 //--
                                                 //   Normal case
                                                 //--
                                                 break;
                                      else
                                                 //--
                                                 // this case error
                                                 //--
                                                 break;
                           }
                }
 }
}

//--------------------------------------------------------------------------
//              Resume WSS routine
//--------------------------------------------------------------------------
void            resumeWSS(void)
{
 WORD           loop;
 BYTE           i;

 wssWR(0x0c,(wss.mode<<6));
 wssWR(0x0a,wss.iControl);

 //----
 // Set audio format
 //----
 wssWR(0x49,(wss.interface&0xfc));              // Playback & Capture are masked
 wssWR(0x48,wss.playFormat);

 for(loop=0;loop<100;loop++)
 {
                if( (inp(cfg.wssBase+4)&0x80) != 0x80 )
                           break;
 }

 if(loop==100)
 {
                // Error Routine
 }

 outp(cfg.wssBase+4,0x08);                     // Modified: May 2nd, 1997

 if( wss.interface&0x08 == 0x08)
 {
                outp(cfg.wssBase+4,0xb);        // Modified: May 2nd, 1997
                for(loop=0;loop<100;loop++)     // Modified: May 2nd, 1997
                           inp(cfg.wssBase+5);
```

```
        for(loop=0;loop<10000;loop++)
        {
                if( (inp(cfg.wssBase+5)&0x20) != 0x20 )        // Modified: May 2nd, 1997
                        break;
        }
}

if(wss.mode)
{
        wssWR(0x5c,wss.recFormat);
        outp(cfg.wssBase+4,0x1c);                        // Modified: May 22nd, 1997

        if( wss.interface&0x08 == 0x08)
        {
                outp(cfg.wssBase,0xb);
                for(loop=0;loop<100;loop++)
                        inp(cfg.wssBase+5);
                for(loop=0;loop<10000;loop++)
                {
                        if( (inp(cfg.wssBase+5)&0x20) != 0x20 )        // modified on May 22,
1997
                                break;
                }
        }

        wssWR(0x10,(wss.dacConfig&0xbf));                // modified on May 22, 1997
        wssWR(0x15,wss.timerUpper);
        wssWR(0x14,wss.timerLower);
}

// modified on May 22, 1997
// Order of setting Base address counter was changed.
// After both playback and recording audio format were set, write base address
// and current address.
wssWR(0x0f,wss.playBaseLower);
wssWR(0x0e,wss.playBaseUpper);
ctrlWR(0x0b,wss.playCurrentLower);
ctrlWR(0x0c,wss.playCurrentUpper);

if(wss.mode)
{
        wssWR(0x1f,wss.recBaseLower);
        wssWR(0x1e,wss.recBaseUpper);
        ctrlWR(0x0d,wss.recCurrentLower);
        ctrlWR(0x0e,wss.recCurrentUpper);
}


//----
// Assert the interrupt pin, if OPL3-SA3 switched suspend
// while interrupt pin was asserted.
//----
if( (wss.mode==0) && ((wss.status&0x01)==0x01) ){
        ctrlWR(0x0f,0x01);
```

```
                // modified May 9th
                ctrlWR(0x0f,0x00);
 }
 else if( (wss.mode==1) && ((wss.status&0x01)==0x01) ){
                ctrlWR(0x0f,(wss.iStatus>>4)&0x07);
                // modified May 9th
                ctrlWR(0x0f,0x00);
                printf("wss Interrupt\n");
                //getch();
 }

 //----
 // Start Playback, Capture, Timer, if these bits ware set.
 //----
 wssWR(0x09,wss.interface);
 if(wss.mode)
                wssWR(0x10,wss.dacConfig);

}

//------------------------------------------------------------------------
//              Resume Power Management state
//------------------------------------------------------------------------
void            resumePowerDown(void)
{
 BYTE           pmreg;

 pmreg = (pm.aSave<<5)|(pm.save<<2)|(pm.down<<1)|(pm.clock);
 ctrlWR(0x01,pmreg);
 ctrlWR(0x12,pm.part1);
 ctrlWR(0x13,pm.part2);
}

//------------------------------------------------------------------------
//              Unmute master volume
//------------------------------------------------------------------------
void            unmuteMaster(void)
{
 ctrlWR(0x07,mix.masterL);
 ctrlWR(0x08,mix.masterR);
}


//------------------------------------------------------------------------
//              Wait for Active
//
//              Loop until WSS becomes active. Check INIT bit goes to "0"
//------------------------------------------------------------------------
void            waitforActive(void)
{
 WORD           loop;

 for(loop=0;loop<100;loop++)
 {
```

```
                if( (inp(cfg.wssBase+4)&0x80) == 0x00)
                        break;
 }

 if(loop==100);
                // WSS is not present

}


//-------------------------------------------------------------------------
//              Resume Control Register
//-------------------------------------------------------------------------
void            resumeCtrl(void)
{
 ctrlWR(0x02,ctrl.system);
 ctrlWR(0x03,ctrl.irqChannel);
 ctrlWR(0x06,ctrl.dmaChannel);
 ctrlWR(0x0a,ctrl.misc);
}


//-------------------------------------------------------------------------
//              Main routine
//-------------------------------------------------------------------------
void            main(void)
{
 FILE           *fp;
 WORD           data;

 if( (fp=fopen("susres.dat","rb")) == NULL )
 {
                printf("Can't open susres.dat\n");
                return;
 }

 fread(resourceData,512,1,fp);
 fread(&cfg,18,1,fp);
 fread(&mix,18,1,fp);
//              modified May 8th
 fread(&sb,8,1,fp);
 fread(&wss,21,1,fp);
 fread(&opl,241,1,fp);
 fread(&pm,6,1,fp);
 fread(&ctrl,5,1,fp);
//              Modified Aug 1st
 fread(scanData,28,1,fp);
 fread(&dmaAState,1,1,fp);
 fread(&dmaBState,1,1,fp);
 fread(&mpuMode,1,1,fp);

 fclose(fp);

 if(resumeConfig())
```

```
{
                printf("OPL3-SA3 was not found or was not activated.\n");
                return;
}

waitforActive();
//              modified May 8th
resumeOPL();
//                              printf("Write back OPL3 Setting\n");
resumeMPU();
//                              printf("Write back MPU Setting\n");
resumeMixer();
//                              printf("Write back Mixer Setting\n");
resumeSBmixer();
//                              printf("Write back SBpro Mixer Setting\n");
resumeCtrl();
//                              printf("Write back Control Register Setting\n");
unmuteMaster();
//                              printf("Unmuted\n");
resumeWSS();
//                              printf("Write back WSS Setting\n");
resumeSB();
//                              printf("Write back SB Setting\n");
currentIndex();
//                              printf("Unmask DMA\n");
unmaskDMA();

}

-
```

GMPPROC2.ASM

```
.386

;-------------------------------------------------------------------------------
;   EXTERN DECLARATION
;-------------------------------------------------------------------------------
EXTRN   WaitTime:NEAR

EXTRN   gbMLExistsFlg:WORD
EXTRN   oplBase:WORD
EXTRN   gbGMPSaveData:BYTE
EXTRN   gbGMPSaveDataCnt:WORD
EXTRN   gbGMPSusOK:BYTE
EXTRN   gbMLProcessorFlg:BYTE
EXTRN   gbMLSusResFlg:BYTE
EXTRN   gbMLPowerFlg:BYTE

proDevIdStr label     byte
 db             "GMP_OPL4", 0, 1eH
proSoftDatStr label     byte
            db         0, 0, 0, 0, 0, 0
proSoftVerChkStr label byte
            db         1,2,2
            db         1,2,3
            db         1,2,4
            db         2,0,0


;----------------------------------------------------------------------------;
;
;    GMPPowerDownBusIn
;
;    DESCRIPTION:
;              Power Down In(at OPL4-ML2 Bus Connect)
;
;    ENTRY:
;
;    EXIT:
;
;    USES:
;
;----------------------------------------------------------------------------;
GMPPowerDownBusIn  proc near

 pushad

 mov        al, 0FDH                  ; GMP COMMAND = 0FDH
 call       GMPWrite
 jc         GMPPowerDownBusIn_Err

 mov        ecx, 30                               ; wait 30ms
 call       WaitTime
 call         GMPReady
 jc           GMPPowerDownBusIn_Err

 popad
 clc
 ret

GMPPowerDownBusIn_Err:
 popad
 stc
 ret

GMPPowerDownBusIn  endp

;-----------------------------------------------------------------------;
;
;   GMPPowerDownBusOut
;
;   DESCRIPTION:
;             Power Down Out(at OPL4-ML2 Bus Connect)
;
;   ENTRY:
;
;   EXIT:
;
;   USES:
;
;-----------------------------------------------------------------------;
GMPPowerDownBusOut            proc near

 pushad

 mov          al, 85H                        ; GMP CONTROL
 call         GMPCtlWrite

 mov          ecx, 100             ; wait 100ms
 call         WaitTime

 mov          al, 05H                        ; GMP CONTROL
 call         GMPCtlWrite

 mov          ecx, 3                         ; wait 3ms
 call         WaitTime

 mov          ax, 0
 mov          dx,oplBase
 add          dx, 7                          ; dx = oplBase + 7
 in           al, dx                         ;read data

 popad
 clc
 ret

GMPPowerDownBusOut_Err:
 popad
 stc
 ret

GMPPowerDownBusOut          endp

;-----------------------------------------------------------------------;
;
;    GMPSuspend
;
;    DESCRIPTION:
;         This function Get GMP Suspend DATA.
;
;    ENTRY:
;
;    EXIT:
;
;    USES:
;         Flags, EAX, EBX, ECX, EDX
;
;-----------------------------------------------------------------------;
GMPSuspend          proc near

 pushad

 mov        dx, [edi.sai_wIOAddressOPL]   ; AdLib base LOAD
 mov        oplBase, dx

 cmp        gbMLExistsFlg, 0      ; ML EXISTS FLG OFF ?
 je         GMPSuspend_err

 mov        DX, 0                           ; DX:SUM CHECK DATA

 mov        al, 0e0H              ; COMMAND
 call       GMPWrite
 jc         GMPSuspend_err

 call       GMPReady
 jc         GMPSuspend_err

 mov        dx,oplBase
 add        dx, 7                           ; dx = oplBase + 7
 in         al, dx
 test       al, 04H                         ; RESP BIT ON ?
 jz         GMPSuspend_err

 call       GMPRead
 jc         GMPSuspend_err
 cmp        al, 0ffH
 jne        GMPSuspend_err

 mov        dl, 0
 call       GMPRead
 jc         GMPSuspend_err
 add         dl, al                            ; SUM CHECK DATA ADD
 mov         bl, al                            ; DATA LENGTH LOW SAVE

 call        GMPRead
 jc          GMPSuspend_err
 add         dl, al                            ; SUM CHECK DATA ADD
 mov         bh, al                            ; BX = DATA LENGTH
 mov         gbGMPSaveDataCnt, bx

 mov         edi, OFFSET32 gbGMPSaveData
 mov         cx, 0                             ; CX = COUNTER

GMPSuspend_loop:
 call        GMPRead
 jc          GMPSuspend_err
 mov         byte ptr [edi], al        ; GMP DATA SAVE
 add         dl, al                            ; SUM CHECK DATA ADD
 inc         edi
 inc         cx
 cmp         cx, bx
 jb          GMPSuspend_loop

 ; SUM CHECK

 call        GMPRead
 jc          GMPSuspend_err
 add         dl, al
 and         dl, 0ffH
 cmp         dl, 0
 jne         GMPSuspend_err

 mov         gbGMPSusOK, 0
 popad
 clc
 ret

GMPSuspend_err:
 mov         gbGMPSusOK, 1
 popad
 stc
 ret

GMPSuspend              endp

;------------------------------------------------------------------------;
;
;    GMPResume
;
;    DESCRIPTION:
;         This function Load GMP Suspend DATA.
;
;    ENTRY:
;
;    EXIT:
```

```
;
;    USES:
;         Flags, EAX, EBX, ECX, EDX
;
;-------------------------------------------------------------------------;
BeginProc GMPResume

 pushad

 mov       dx, [edi.sai_wIOAddressOPL]  ; AdLib base LOAD
 mov       oplBase, dx

 cmp       gbMLExistsFlg, 0      ; ML EXISTS FLG OFF ?
 je        GMPResume_err

 cmp       gbGMPSusOK, 1                  ; Suspend ERRORED
 je        GMPResume_err
 mov       cx, 0

GMPResume_loop:
 call      GMPRead
 jc        GMPResume_next
 inc       cx
 cmp       cx, 32000
 je        GMPResume_err
 jmp       GMPResume_loop

GMPResume_next:
 mov       al, 0e1H              ; GMP COMMAND = E1H
 call      GMPWrite
 jc        GMPResume_err

 mov       al, 0                          ; SUB COMMAND
 call      GMPWrite
 jc        GMPResume_err

 mov       bl, 0                          ; BX : SUM CHECK DATA
 mov       edi, OFFSET32 gbGMPSaveData
 mov       cx, 0                          ; LOOP COUNTER

GMPResume_encode_loop:
 mov       al, byte ptr [edi]

 inc       edi
 inc       cx
 cmp       gbGMPSaveDataCnt, cx
 je        GMPResume_end

; GMP DATA ENCODE
 cmp       al, 7eH
 jae       GMPResume_encode1
 call      GMPWrite              ; GMP DATA WRITE
 jc        GMPResume_err
 add       bl, al                         ; SUM CHECK DATA ADD
 jmp        GMPResume_encode_loop

GMPResume_encode1:
 cmp        al, 80H
 jae        GMPResume_encode2
 mov        ah, al
 mov        al, 7eH
 call       GMPWrite            ; GMP DATA WRITE (7EH)
 jc         GMPResume_err
 add        bl, al                          ; SUM CHECK DATA ADD
 mov        al, ah
 call       GMPWrite            ; GMP DATA WRITE
 jc         GMPResume_err
 add        bl, al                          ; SUM CHECK DATA ADD
 jmp        GMPResume_encode_loop

GMPResume_encode2:
 mov        ah, al
 mov        al, 7fH
 call       GMPWrite            ; GMP DATA WRITE (7FH)
 jc         GMPResume_err
 add        bl, al                          ; SUM CHECK DATA ADD
 mov        al, ah
 and        al, 7fH
 call       GMPWrite            ; GMP DATA WRITE
 jc         GMPResume_err
 add        bl, al                          ; SUM CHECK DATA ADD
 jmp        GMPResume_encode_loop

GMPResume_end:
 not        bl
 and        bl, 7fH
 mov        al, bl
 call       GMPWrite            ; CHECK SUM   DATA WRITE
 jc         GMPResume_err

 clc
 popad
 ret

GMPResume_err:
 stc
 popad
 ret

GMPResume           endp

;--------------------------------------------------------------------------;
;
;   GMPWrite
;
;   DESCRIPTION:
;       This function write to GMP Reg.
;
```

```
;    ENTRY:
;    AL = data to be written
;    oplBase = OPL REG BASE
;
;    EXIT:
;         Carry flag is set if configuration register does not exist
;
;    USES:
;
;------------------------------------------------------------------------;
GMPWrite  proc near

 call         GMPReady
 jc           GMPWrite_err

 push dx
 mov          dx,oplBase
 add          dx, 6
 out          dx, al ;write data

 pop          dx

 clc
 ret

GMPWrite_err:
 stc
 ret

GMPWrite  endp

;------------------------------------------------------------------------;
;
;    GMPCtlWrite
;
;    DESCRIPTION:
;         This function write to GMP Control Reg.
;
;    ENTRY:
;    AL = data to be written
;    oplBase = OPL REG BASE
;
;    EXIT:
;
;    USES:
;
;------------------------------------------------------------------------;
GMPCtlWrite              proc near

 push         dx

 mov          dx,oplBase    ; dx = GMP CONTROL
 add          dx, 7
 out          dx, al ;write data
 pop        dx
 clc
 ret

GMPCtlWrite         endp

;-------------------------------------------------------------------------;
;
;    GMPRead
;
;    DESCRIPTION:
;         This function read to GMP Reg.
;
;    ENTRY:
;    AL = data to be written
;    oplBase = OPL REG BASE
;
;    EXIT:
;    AL = read data
;
;    USES:
;
;-------------------------------------------------------------------------;
GMPRead             proc near

 push       cx
 push       dx

 ;Data read sometimes fails if there is no wait time.
 ;This phenomenon is not described in the hardware document.
 mov ecx, 100
 call WaitTime

 mov        dx,oplBase    ; dx = GMP STATUS
 add        dx, 7
 mov        cx, 0     ; cx = LOOP CNT

GMPRead_loop:
 in         al, dx ;read data
 test       al, 01H
 jnz        GMPRead_next
 inc        cx
 cmp        cx, 60000
 je         GMPRead_err
 test       al, 04H
 jnz        GMPRead_err
 jmp        GMPRead_loop

GMPRead_next:
 call       GMPAlive
 jc         GMPRead_err
 mov        dx,oplBase
 add        dx, 7
 in          al, dx

 mov         cx, 0

GMPRead_loop2:
 inc         cx
 cmp         cx, 1100
 jne         GMPRead_loop2

 pop         dx
 pop         cx
 clc
 ret

GMPRead_err:
 pop         dx
 pop         cx
 stc
 ret

GMPRead                 endp


;--------------------------------------------------------------------------;
;
;   GMPReady
;
;   DESCRIPTION:
;       This function GMP busy bit check.
;
;   ENTRY:
;
;   EXIT:
;       Carry flag is set if configuration register does not exist
;
;   USES:
;
;--------------------------------------------------------------------------;
GMPReady  proc near

 push        cx
 push        dx
 push        ax

 mov         cx, 0
 mov         dx, oplBase
 add         dx, 7                           ; dx = oplBase + 7

GMPReady_loop:
 mov         eax, 0
 in          al, dx
 test        al, 02H
 jz          GMPReady_end
 push        ecx
 mov        ecx, 1                       ; wait 3ms
 call       WaitTime
 pop        ecx
 inc        cx
 cmp        cx, 100
 je         GMPReady_error
 jmp        GMPReady_loop

GMPReady_error:
 pop        ax
 pop        dx
 pop        cx

 stc
 ret

GMPReady_end:
 pop        ax
 pop        dx
 pop        cx

 clc
 ret

GMPReady  endp

;-------------------------------------------------------------------------;
;
;    GMPAlive
;
;    DESCRIPTION:
;         This function GMP Alive check.
;
;    ENTRY:
;
;    EXIT:
;         Carry flag is set if configuration register does not exist
;
;    USES:
;
;-------------------------------------------------------------------------;
GMPAlive   proc near

 mov        al, 0feH
 call GMPWrite
 call GMPReady
 ret

GMPAlive   endp
```

ALL_NOTE_OFF.ASM

```
;---------------------------------------------------------------------------
; MPU401_AllHoldOFF
;
; DESCRIPTION:
;   Send All Hold OFF message.
;
; ENTRY:
;   EDI = pointer to SASNDSYSINFO
;
; EXIT:
;   Write MIDI Message "Bn 40 00"(n = 0...F)
;
; USES:
;   FLAGS, eax, ebx
;---------------------------------------------------------------------------
BeginProc MPU401_AllHoldOFF
 push        eax
 push        ebx

 mov         bl, 0B0h

Loop_AllHoldOFF:
 mov         al, bl
 call        MPU401_Send_Message
 mov         al, 40h
 call        MPU401_Send_Message
 mov         al, 00
 call        MPU401_Send_Message
 inc         bl
 cmp         bl, 0bfh
 je          End_AllHoldOFF
 jmp         Loop_AllHoldOFF
End_AllHoldOFF:

 ;           All Hold2 off
 mov         bl, 0B0h

Loop_AllHold2OFF:
 mov         al, bl
 call        MPU401_Send_Message
 mov         al, 42h
 call        MPU401_Send_Message
 mov         al, 00
 call        MPU401_Send_Message
 inc         bl
 cmp         bl, 0bfh
 je          End_AllHold2OFF
 jmp         Loop_AllHold2OFF
End_AllHold2OFF:
 pop         ebx
 pop        eax
 ret

EndProc     MPU401_AllHoldOFF

;----------------------------------------------------------------------------
; MPU401_AllNoteOFF
;
; DESCRIPTION:
;   Send All Note OFF message.
;
; ENTRY:
;
; EXIT:
;   Write MIDI Message "Bn 7B 00"(n = 0...F)
;
; USES:
;   FLAGS, eax, ebx
;----------------------------------------------------------------------------
BeginProc MPU401_AllNoteOFF
 push       eax
 push       ebx

 mov        bl, 0B0h

Loop_AllNoteOFF:
 mov        al, bl
 call       MPU401_Send_Message
 mov        al, 7Bh
 call       MPU401_Send_Message
 mov        al, 00
 call       MPU401_Send_Message
 inc        bl
 cmp        bl, 0BFh
 je         End_AllNoteOFF
 jmp        Loop_AllNoteOFF
End_AllNoteOFF:
 pop        ebx
 pop        eax
 ret

EndProc     MPU401_AllNoteOFF

;----------------------------------------------------------------------------
;           MPU401_Send_Message
;
;           DESCRIPTION:
;                       Send message through MPU401 I/F.
;           ENTRY:
;                       al = message
;           EXIT:
;
; USES:
;           FLAGS
```

```
;-------------------------------------------------------------------------
BeginProc MPU401_Send_Message

 push     edx
 push     eax

 movzx edx, wIOAddressMPU
 inc      edx
MPU401_Send_Busy:
          ; check FIFO busy flag
 in       al, dx
 test     al, 40h
 jnz      short MPU401_Send_Busy

 pop      eax
 dec      edx
 out      dx, al     ; send data

 pop      edx
 ret

EndProc MPU401_Send_Message
```

This page is intentionally blank.

OL3-SA3 Suspend/Resume BIOS assembler code

1. File structure

Sample codes contain these files.

SUSTEST.ASM	:	Test program for Suspend.	: Page 51
RESTEST.ASM	:	Test program for Resume.	: Page 54
SUSRES.INC	:	code for macro difinition	: Page 57
PNPISA.INC	:	code for getting resources	: Page 59
SUSPEND.INC	:	code for Suspend.	: Page 63
RESUME.INC	:	code for Resume.	: Page 78
DATA.INC	:	Definition of Work area	: page 95

Test program includes these files below when assembling. Please put all files at the same directory.

SUSTEST.ASM	:	SUSRES.INC, PNPISA.INC, SUSPEND.INC, DATA.INC
RESTEST.ASM	:	SUSRES.INC, PNPISA.INC, RESUME.INC, DATA.INC

2. Executing programs

This test program executes Suspend/Resume under pure MS-DOS environment.
This program does not configure the Stack Segment, so that please make execution file of "COM" , in case of MASM using EXE2BIN after linking, in case of TASM use /t option.

When executes SUSTEST.COM, it returns a return code(00H) and output 346 bytes file which name is "TEST.DAT" to the directory where executed.
In case of Suspend error, returns the codes below and does not make output file.

Return code 01H	:	fail at SoundBlaster suspend.
Return code 02H	:	fail at writing TEST.DAT.

When executes RESTEST.COM, it reads "TEST.DAT" and goes through resuming.
If terminates normally, it returns return code(00H). In case of Resume error, it returns the codes below.

Return code 01H	:	fail at reading TEST.DAT.
Return code 02H	:	fail at activating WSS.
Return code 03H	:	fail at MPU Resume.
Return code 04H	:	fail at WSS Resume.

3. Constant

The constants that is defined in the code is described as below.

PnP_ADDR = 0279H	PnP-ISA address port
PnP_DATA = 0A79H	PnP-ISA write data port
PnP_RDDA = 0203H	PnP-ISA read data port
Slot_Size = 18	OPL3 Slot numbers
Array_Size = 9	OPL3 Channel numbers

4. Macro

Macro is defined as below.

GetRsrcByte Index, WorkBuf
    Read the value of the PnP-ISA Configuration register which is specified by Index, and write it to [WorkBuf].

GetRsrcWord Index, WorkBuf
    Read the value of the PnP-ISA Configuration register which is specified by Index and Index+1m, and write them to [WorkBuf] and [WorkBuf+1].

PutRsrcByte Index, WorkBuf
    Write the value of [WorkBuf] to the PnP-ISA Configuration register which is specified by Index.

PutRsrcWord Index, WorkBuf
    Write the value of [WorkBuf] and [WorkBuf+1] to the PnP-ISA Configuration register which is specified by Index and Index+1.

GetReg Index, WorkBuf
    Read the value of the register which is specified by Index, and write it to [WorkBuf]. Please specify the I/O port address at DX.

PutReg Index, WorkBuf
    Write the value of [WorkBuf] to the register which is specified by Index. Please specify the I/O port address at DX.

GetOPL Index, WorkBuf
    Read the value of the OPL3 data register which is specified by Index, and write it to [WorkBuf]. Please specify BL to register array 0 or 1, and specify DX to AdlibBase.

PutOPL Index, WorkBuf
    Write the value of [WorkBuf] to the OPL3 data register which is specified by Index. Please specify BL to register array 0 or 1, and specify DX to AdlibBase.

5. Carry Flag of return

These below routines returns with carry flag when process is failed.
Checking carry flags after these routine, Error recovery routine can be added.

SuspendSB, WaitForActive, ResumeMPU, ResumeWSS, WssWait

SUSTEST.ASM

```
;--------------------------------------------------------------------
;           Suspend Routine for OPL3-SA3 (YMF715)
;
;           Copyright (C) 1997, YAMAHA Corporation.
;           All rights reserved.
;
;           History:    Version 0.1         May   9th, 1997
;                       Version 0.2         May 27th, 1997
;                       Version 0.3         May 29th, 1997
;                       Version 0.4         Jun   7th, 1997
;
;--------------------------------------------------------------------

            INCLUDE SUSRES.INC

            code        segment
            assume      cs:code, ds:code, ss:code
            ORG 100H
.386

suspend:

            call        MaskDMA
            call        GetLogDev
            call        GetIndex
            call        SuspendMixer
            call        SuspendSbMixer
            call        SuspendOPL
            call        SuspendSB
            jc          Suspend_Error1
            call        SuspendMPU
            call        SuspendWSS
            call        SuspendCtrl
            call        OplDataWrite
            jc          Suspend_Error2

            mov         ah, 4Ch
            mov         al, 00h             ; Error_Code = 00h
            int         21h                 ; terminate program

Suspend_Error1:
            mov         ah, 4Ch
            mov         al, 01h             ; Error_Code = 01h
            int         21h                 ; terminate program

Suspend_Error2:
            mov         ah, 4Ch
            mov         al, 02h             ; Error_Code = 02h
            int         21h                 ; terminate program
```

```
;---------------------------------------------------------------------------;
;               OplDataWrite
;
;    ENTRY:
;                               Nothing.
;    EXIT:
;                               Carry = 0 : Suceed.
;                               Carry = 1 : Error.
;---------------------------------------------------------------------------;
OplDataWrite            proc near

                push        ax
                push        bx
                push        cx
                push        dx

                mov         ah, 3Ch
                mov         dx, offset Susres_FileName
                mov         cx, 20h
                int         21h                         ; Create File
                jc          OplDataWrite_Error

                mov         ax, 3D01h
                int         21h                         ; Open File
                jc          OplDataWrite_Error

                mov         bx, ax
                mov         cx, Opl_Size
                mov         dx, offset ResourceData
                mov         ah, 40h
                int         21h                         ; Data Write
                jc          OplDataWrite_Error

                mov         ah, 3Eh
                int         21h                         ; File Close
                jc          OplDataWrite_Error

                pop         dx
                pop         cx
                pop         bx
                pop         ax
                clc
                ret

OplDataWrite_Error:
                pop         dx
                pop         cx
                pop         bx
                pop         ax
                stc
                ret

OplDataWrite            endp
```

```
INCLUDE  PNPISA.INC
INCLUDE  SUSPEND.INC

INCLUDE DATA.INC
SusRes_FileName     db          "test.dat",00h

code        ends
end         suspend
```

-

RESTEST.ASM

```
;-----------------------------------------------------------------------
;           Resume Routine for OPL3-SA3 (YMF715)
;
;           Copyright (C) 1997, YAMAHA Corporation.
;           All rights reserved.
;
;           Assembler: Turbo Assembler Version 4.0
;           History:   Version 0.1          May   9th, 1997
;           History:   Version 0.2          May 27th, 1997
;           History:   Version 0.3          May 29th, 1997
;           History:   Version 0.4          Jun   7th, 1997
;
;-----------------------------------------------------------------------

            INCLUDE SUSRES.INC

            code        segment
            assume      cs:code, ds:code, ss:code
            ORG 100H
.386

resume:
            call        OplDataRead
            jc          Resume_Error1
            call        SetLogDev
            call        WaitForActive
            jc          Resume_Error2
            call        ResumeMixer
            call        ResumeSbMixer
            call        ResumeOPL
            call        ResumeSB
            call        ResumeMPU
            jc          Resume_Error3
            call        ResumeWSS
            jc          Resume_Error4
            call        ResumeCtrl
            call        UnmaskDMA
            call        UnmuteMaster
            call        CurrentIndex

            mov         ah, 4Ch
            mov         al, 00h             ; Error_Code = 00h
            int         21h                 ; Terminate Program

Resume_Error1:
            mov         ah, 4Ch
            mov         al, 01h             ; Error_Code = 01h
            int         21h                 ; Terminate Program

Resume_Error2:
            mov         ah, 4Ch
            mov     al, 02h                 ; Error_Code = 02h
            int     21h                     ; Terminate Program

Resume_Error3:
            mov     ah, 4Ch
            mov     al, 03h                 ; Error_Code = 03h
            int     21h                     ; Terminate Program

Resume_Error4:
            mov     ah, 4Ch
            mov     al, 04h                 ; Error_Code = 04h
            int     21h                     ; Terminate Program

;-----------------------------------------------------------------------;
;           OplDataRead
;
;   ENTRY:
;                       Nothing.
;   EXIT:
;                       Carry = 0 : Suceed.
;                       Carry = 1 : Error.
;-----------------------------------------------------------------------;
OplDataRead             proc near

            push    ax
            push    bx
            push    cx
            push    dx

            mov     dx, offset Susres_FileName
            mov     ah, 3Dh
            int     21h
            jc      OplDataRead_Error

            mov     bx, ax
            mov     cx, Opl_Size
            mov     dx, offset ResourceData
            mov     ah, 3Fh
            int     21h
            jc      OplDataRead_Error

            mov     ah, 3Eh
            int     21h
            jc      OplDataRead_Error

            pop     dx
            pop     cx
            pop     bx
            pop     ax
            clc
            ret

OplDataRead_Error:
            pop     dx
            pop         cx
            pop         bx
            pop         ax
            stc
            ret

OplDataRead             endp


            INCLUDE  PNPISA.INC
            INCLUDE  RESUME.INC

            INCLUDE DATA.INC
            SusRes_FileName     db          "test.dat",00h

            code        ends
            end         resume
-
```

SUSRES.INC

```
PnP_ADDR EQU        0279h
PnP_DATA EQU        0A79h
PnP_RDDA EQU        0203h
Slot_Size    EQU        18
Array_Size  EQU        9

;*******************************************************************
GetRsrcByte             macro       Index, WorkBuf
                mov         ah, Index
                call        PnP_Read
                mov         [WorkBuf], al
                endm
;*******************************************************************
GetRsrcWord             macro       Index, WorkBuf
                mov         ah, Index
                call        PnP_Read
                mov         dh, al
                inc         ah
                call        PnP_Read
                mov         dl, al
                mov         [WorkBuf], dx
                endm
;*******************************************************************
GetOPL                  macro       Index, Workbuf
                ; BL = Register Array (0 or 1)
                ; DX = Adlib Base

                mov         ah, Index
                call        GetOplReg
                mov         [WorkBuf], al
                endm
;*******************************************************************
PutRsrcByte             macro       Index, WorkBuf
                mov         ah, Index
                mov         al, [WorkBuf]
                call        PnP_Write
                endm
;*******************************************************************
PutRsrcWord             macro       Index, WorkBuf
                mov         ah, Index
                mov         dx, [WorkBuf]
                mov         al, dh
                call        PnP_Write
                mov         al, dl
                inc         ah
                call        PnP_Write
                endm
;*******************************************************************
GetReg          macro       Index, WorkBuf
                ; DX = PORT ADDRESS
```

```
        mov     ah, Index
        call    Reg_Read
        mov     [WorkBuf], al
        endm
;*********************************************************************
PutReg  macro   Index, WorkBuf
        ; DX = PORT ADDRESS

        mov     ah, Index
        mov     al, [WorkBuf]
        call    Reg_Write
        endm
;*********************************************************************
PutOPL  macro   Index, Workbuf
        ; BL = Register Array (0 or 1)
        ; DX = Adlib Base

        mov     ah, Index
        mov     al, [WorkBuf]
        call    PutOplReg
        endm
```

PNPISA.INC

```
;----------------------------------------------------------------------;
;               PnP_Write
;
;     ENTRY:
;                                         AL = Write data
;                                         AH = Write index
;
;     EXIT:
;                                         Nothing.
;----------------------------------------------------------------------;
PnP_Write  proc         near

           push         ax
           push         dx

           mov          dx, PnP_ADDR
           xchg         al, ah                 ; AH = Data, AL = Index
           out          dx, al                 ; Write Index

           mov          dx, PnP_DATA
           xchg         al, ah                 ; AH = Index, AL = Data
           out          dx, al                 ;write data


           pop          dx
           pop          ax
           ret

PnP_Write  endp

;----------------------------------------------------------------------;
;               PnP_Read
;
;     ENTRY:
;                                         AH = Read index
;
;     EXIT:
;                                         AL = Read data
;----------------------------------------------------------------------;
PnP_Read   proc         near

           push         dx

           mov          dx, PnP_ADDR
           mov          al, ah
           out          dx, al                 ;write index

           mov          dx, PnP_RDDA
           in           al, dx                 ;read data
```

```
                pop         dx
                ret

PnP_Read    endp

;-----------------------------------------------------------------------;
;               Reg_Write
;
;   ENTRY:
;                                   AH = Write index
;                                   AL = Write data
;                                   DX = I/O address
;
;   EXIT:
;                                   Nothing.
;-----------------------------------------------------------------------;
Reg_Write   proc        near

                push        ax
                push        cx
                push        dx

                xchg        al, ah              ; AH = Data AL= Index
                out         dx, al              ; Write Index
                inc         dx
                xchg        al, ah              ; AH = Index, AL = Data
                out         dx, al              ; Write Data
                mov         cx, 100
                call        WaitTime

                pop         dx
                pop         cx
                pop         ax
                ret

Reg_Write   endp

;-----------------------------------------------------------------------;
;               Reg_Read
;
;   ENTRY:
;                                   AH = Read index
;                                   DX = I/O Address
;
;   EXIT:
;                                   AL = Read data
;-----------------------------------------------------------------------;
Reg_Read    proc        near

                push        dx

                mov         al, ah
                out         dx, al              ;write index
```

```
          inc       dx
          in        al, dx              ;read data

          pop       dx
          ret

Reg_Read  endp

;-----------------------------------------------------------------------;
;         SendYAMAHAKey
;
;   ENTRY:
;                             Nothing.
;
;   EXIT:
;                             Nothing.
;-----------------------------------------------------------------------;
SendYamahaKey       proc      near

          push      ax
          push      bx
          push      cx
          push      dx

          mov       cl, YamahaKey_Size
          mov       dx, PnP_ADDR
          mov       bx, offset YamahaKey

LoopWriteKey:
          mov       al, [bx]
          out       dx, al
          inc       bx
          dec       cl
          jnz       LoopWriteKey

          pop       dx
          pop       cx
          pop       bx
          pop       ax
          ret

SendYAMAHAKey       endp

;-----------------------------------------------------------------------;
;         WaitTime
;
;   ENTRY:          CX = Wait Counter
;                   WaitTime: 60ns * CX
;
;   EXIT:
;                   Nothing
;-----------------------------------------------------------------------;
WaitTime  proc near
```

```
                push            cx

WaitTime_Loop:
                dec             cx
                jnz             WaitTime_Loop

                pop             cx
                ret

WaitTime        endp
```

SUSPEND.INC

```
;---------------------------------------------------------------------------;
;               GetLogDev
;
;     ENTRY:
;                                   Nothing.
;
;     EXIT:
;                                   Nothing.
;---------------------------------------------------------------------------;
GetLogDev               proc        near

                push        ax
                push        bx
                push        cx
                push        dx

                mov         ah, 02h                             ; Wait for key state
                mov         al, 02h
                call        PnP_Write
                call        SendYamahaKey

                mov         ah, 03h
                mov         al, 81h
                call        PnP_Write               ; Write [CSN]

                xor         ah, ah
                mov         al, 80h
                call        PnP_Write               ; Force to set 203h ReadPort

                mov         ah, 06h
                call        PnP_Read

                ; LDN=0 <SA2 Sound System>

                mov         ah, 07h
                mov         al, 00h
                call        PnP_Write                           ; Set LDN=0

                ; Read IO port address

                GetRsrcWord 60h, sbBase
                GetRsrcWord 62h, wssBase
                GetRsrcWord 64h, adlibBase
                GetRsrcWord 66h, mpuBase
                GetRsrcWord 68h, ctrlBase
                GetRsrcByte 70h, IRQ_A
                GetRsrcByte 72h, IRQ_B
                GetRsrcByte 74h, DMA_A
```

```
            GetRsrcByte 75h, DMA_B
            GetRsrcByte 30h, Active0

            ; LDN=1 <Joy Stick>

            mov ah, 07h
            mov al, 01h
            call        PnP_Write               ; Set LDN=1

            ; Read IO port address

            GetRsrcWord 60h, JoyBase
            GetRsrcByte 30h, Active1

            mov         ah, 02h
            mov         al, 02h
            call        PnP_Write               ; Wait for key state

            pop         dx
            pop         cx
            pop         bx
            pop         ax
            ret

GetLogDev               endp

;---------------------------------------------------------------------------;
;           MaskDMA
;
;   ENTRY:
;                       Nothing.
;
;   EXIT:
;                       Nothing.
;---------------------------------------------------------------------------;
MaskDMA                 proc        near

            push        ax
            push        cx

            in          al, 0Fh
            not         al
            mov         ah, al
            mov         cl, DMA_A
            shr         al, cl
            and         al, 01h
            mov         DMA_A_State, al
            jz          MaskDMA_Skip

            ; Mask DMA_A channel

            mov         al, cl
            or          al, 04h
            out         0Ah, al
```

```
MaskDMA_Skip:
                mov     al, ah
                mov     cl, DMA_B
                shr     al, cl
                and     al, 01h
                mov     DMA_B_State, al
                jz      MaskDMA_End

                ; Mask DMA_B channel

                mov     al, cl
                or      al, 04h
                out     0Ah, al

MaskDMA_End:
                pop     cx
                pop     ax
                ret

MaskDMA                 endp

;------------------------------------------------------------------------;
;               GetIndex
;
;    ENTRY:
;                       Nothing.
;
;    EXIT:
;                       Nothing.
;------------------------------------------------------------------------;
GetIndex        proc    near

                push    dx

                mov     dx, WssBase
                add     dx, 04h
                in      al, dx
                mov     WssIndex, al

                mov     dx, CtrlBase
                in      al, dx
                mov     CtrlIndex, al

                pop     dx
                ret

getIndex        endp

;------------------------------------------------------------------------;
;               SuspendMixer
;
;    ENTRY:
;                       Nothing.
```

```
;
;   EXIT:
;                       Nothing.
;   USES:
;                       CX, DX = I/O Port Address
;-----------------------------------------------------------------------;
SuspendMixer                    proc        near

            push        ax
            push        cx
            push        dx

            mov         dx, CtrlBase
            GetReg      07h, VolumeL
            GetReg      08h, VolumeR

            mov         ah, 07h
            mov         al, VolumeL
            or          al, 80h                 ; Mute Left Channel
            call        Reg_Write

            mov         ah, 08h
            mov         al, VolumeR
            or          al, 80h                 ; Mute Right Channel
            call        Reg_Write

            xchg        cx, dx
            mov         dx, WssBase
            add         dx, 4                   ; CX = CtrlBase, DX = Wssbase+4
            mov         ah, 0Ch
            call        Reg_Read
            and         al, 40h
            mov         WssMode, al

            GetReg      00h, InputR
            GetReg      01h, InputL
            GetReg      02h, Aux1L
            GetReg      03h, Aux1R
            GetReg      04h, Aux2L
            GetReg      05h, Aux2R
            GetReg      06h, WaveL
            GetReg      07h, WaveR

            mov         al, WssMode
            or          al, al
            jz          SuspendMixer_Skip

            GetReg      12h, LineL
            GetReg      13h, LineR
            GetReg      1Ah, Mono

SuspendMixer_Skip:

            mov         dx, cx                  ; DX = CtrlBase
            GetReg    09h, Mic
            GetReg    14h, Wide
            GetReg    15h, Bass
            GetReg    16h, Treble
            GetReg    17h, HVInt

            pop       dx
            pop       cx
            pop       ax
            ret

SuspendMixer          endp

;------------------------------------------------------------------------;
;           SuspendOPL
;
;   ENTRY:
;                     Nothing.
;
;   EXIT:
;                     Nothing.
;   USES:
;                     BL = OPL3 Register Array ( 0 or 1 )
;                     DX = Adlib base Address
;------------------------------------------------------------------------;
SuspendOPL            proc        near

            push      ax
            push      bx
            push      dx

            mov       dx, AdlibBase
            mov       bl, 1                         ; Register Array : 1
            GetOPL    05h, Opl3Mode

            xor       bl, bl                        ; Register Array : 0
            GetOPL    08h, Nts
            GetOPL    02h, Opl3Timer1
            GetOPL    03h, Opl3Timer2
            GetOPL    04h, TimerCtrl
            call      ReadOplArray
            GetOPL    0BDh, Rhythm

            mov       al, Opl3Mode
            rcr       al, 1
            jnc       SusOpl3_End

            mov       bl, 1                         ; Register Array : 1
            GetOPL    04h, Opl3Connect
            call      ReadOplArray

SusOpl3_End:
            pop       dx
            pop       bx
```

```
		pop	ax
		ret

SuspendOPL		endp

;-----------------------------------------------------------------------;
;		ReadOplArray
;
;	ENTRY:
;				BL = Register Array ( 0 or 1 )
;				DX = Adlib Base
;
;	EXIT:
;				Nothing.
;
;	USES:
;				AH = OPL3 Register Index Base
;				BL = OPL3 Register Array ( 0 or 1 )
;				CX = Counter
;				DX = Adlib base Address
;				SI = Address Offset
;				DI = OPL3 Store Address
;
;-----------------------------------------------------------------------;
ReadOplArray		Proc near

		push	ax
		push	bx
		push	cx
		push	dx
		push	di
		push	si

		mov	ax, Slot_Size
		mov	cx, ax
		and	bl, 01h
		mul	bl
		mov	si, ax			; SI = 0 (BL=0) or Slot_Size (BL=1)

		; Get Mult

		mov	di, offset Mult0
		add	di, si
		mov	ah, 20h
		call	ReadOplSlot

		; Get Tl0

		mov	di, offset Tl0
		add	di, si
		mov	ah, 40h
		call	ReadOplSlot

		; Get Ad
```

```
        mov     di, offset Ad0
        add     di, si
        mov     ah, 60h
        call    ReadOplSlot

        ; Get Sr

        mov     di, offset Sr0
        add     di, si
        mov     ah, 80h
        call    ReadOplSlot

        ; Get Ws

        mov     di, offset Ws0
        add     di, si
        mov     ah, 0E0h
        call    ReadOplSlot

        mov     ax, Array_Size
        mov     cx, ax
        mul     bl
        mov     si, ax                  ; SI = 0 (BL=0) or Array_Size (BL=1)

        ; Get Fnum
        mov     di, offset Fnum0
        add     di, si
        mov     ah, 0A0h
        call    ReadOplStatus

        ; Get Block
        mov     di, offset Block0
        add     di, si
        mov     ah, 0B0h
        call    ReadOplStatus

        ; Get Fb
        mov     di, offset Fb0
        add     di, si
        mov     ah, 0C0h
        call    ReadOplStatus

        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret

ReadOplArray    endp
```

```
;---------------------------------------------------------------------------;
;               ReadOplSlot
;
;    ENTRY:
;                           AH = Index Base
;                           BL = Register Array ( 0 or 1 )
;                           CX = Counter
;                           DX = Adlib Base Address
;                           DI = Store Address Base
;
;    EXIT:
;                           Nothing.
;
;    USES:
;                           AH = OPl3 Index
;                           BH = Index Base(Stored)
;                           BL = OPL3 Register Array
;                           SI = Opl3Slot Address
;                           DX = Adlib base Address
;---------------------------------------------------------------------------;
ReadOplSlot             Proc        Near

                push        ax
                push        bx
                push        cx
                push        dx
                push        si
                push        di

                mov         si, offset Opl3SlotNo
                mov         bh, ah              ; Store Index base

ReadOplSlot_Loop:
                mov         ah, [si]
                add         ah, bh              ; ah = Index Base + Index Offset
                call        GetOplReg
                mov         [di], al
                inc         si
                inc         di
                dec         cx
                jnz         ReadOplSlot_Loop

                pop         di
                pop         si
                pop         dx
                pop         cx
                pop         bx
                pop         ax

                ret

ReadOplSlot             endp

;---------------------------------------------------------------------------;
```

```
;               ReadOplStatus
;
;    ENTRY:
;                               AH = Index Base
;                               CX = Counter (Index Offset)
;                               DX = Adlib Base Address
;                               DI = Store Address Base
;
;   EXIT:
;                               Nothing.
;
;   USES:
;                               BL = OPL3 Register Array ( 0 or 1 )
;                               DX = Adlib Base address
;---------------------------------------------------------------------------;
ReadOplStatus                   Proc            Near

                push            ax
                push            cx
                push            di

ReadOplStatus_Loop:
                call            GetOplReg
                mov             [di], al
                inc             ah
                inc             di
                dec             cx
                jnz             ReadOplStatus_Loop

                pop             di
                pop             cx
                pop             ax

                ret

ReadOplStatus                   endp

;---------------------------------------------------------------------------;
;               GetOplReg
;
;   ENTRY:
;                               AH = Read Index
;                               BL = Register Array ( 0 or 1 )
;                               DX = AdlibBase Base Address
;
;   EXIT:
;                               AL = Read Data
;---------------------------------------------------------------------------;
GetOplReg Proc                  Near

                push            bx
                push            cx
                push            dx
                rcr        bl, 1
                jnc        GetOplReg_Skip         ;When Array=1, Port Address=AdlibBase+2
                inc        dx
                inc        dx

GetOplReg_Skip:
                call       Reg_Read
                mov        cx, 1
                call       WaitTime

                pop        dx
                pop        cx
                pop        bx
                ret

GetOplReg endp

;-------------------------------------------------------------------------;
;               SuspendSB
;
;    ENTRY:
;                          Nothing
;
;    EXIT:
;                          Carry = 0 : Suceeded.
;                          Carry = 1 : Error.
;-------------------------------------------------------------------------;
SuspendSB proc near

                push       ax
                push       bx
                push       cx
                push       dx

                mov        dx, CtrlBase
                mov        ax, 1001h              ; Index=10h, Data=01h
                call       Reg_Write              ; Set SBPDR
                mov        cx, 100h

SusSB_Loop1:
                call       Reg_Read
                and        al, 80h
                jnz        SusSB_Loop1Exit
                dec        cx
                jnz        SusSB_Loop1

                ; SusSB_Error

                pop        dx
                pop        cx
                pop        bx
                pop        ax
                stc
                ret
```

```
SusSB_Loop1Exit:
        mov     ax, 100Dh  ; Index=10h, Data=0Dh
        call    Reg_Write
        mov     bx, offset SbScanData
        mov     cl, 27

SusSB_Loop2:
        mov     ch, 8

SusSB_Loop3:
        mov     ax, 100Fh
        call    Reg_Write
        mov     al, 0Dh
        call    Reg_Write
        dec     ch
        jnz     SusSB_Loop3

        mov     ah, 11h
        call    Reg_Read
        mov     [bx], al
        inc     bx
        dec     cl
        jnz     SusSB_Loop2

        mov     ax, 100Fh
        call    Reg_Write
        mov     al, 0Dh
        call    Reg_Write
        mov     al, 0Fh
        call    Reg_Write
        mov     al, 0Dh
        call    Reg_Write

        mov     ah, 11h
        call    Reg_Read
        mov     cl, 6
        sal     al, cl
        mov     [bx], al

        mov     ax, 1001h   ; Index =10h, Data = 01h
        call    Reg_Write

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        clc
        ret

SuspendSB endp

;---------------------------------------------------------------------------;
;       SuspendCtrl
```

```
;
;    ENTRY:
;                        Nothing.
;
;    EXIT:
;                        Nothing.
;
;    USES:
;                        DX = Ctrl Base address
;---------------------------------------------------------------------------;
SuspendCtrl              proc near

              push       ax
              push       dx

              mov        dx, CtrlBase

              GetReg     02h, CtrlSystem
              GetReg     03h, IRQchannel
              GetReg     06h, DMAchannel
              GetReg     0Ah, CtrlMisc

              pop        ax
              pop        dx
              ret

SuspendCtrl              endp

;---------------------------------------------------------------------------;
;             SuspendMPU
;
;    ENTRY:
;                        Nothing.
;
;    EXIT:
;                        Nothing.
;---------------------------------------------------------------------------;
SuspendMPU               proc near

              push       ax
              push       cx
              push       dx

              mov        dx, MpuBase
              inc        dx
              mov        al, 0FFh
              out        dx, al
              in         al, dx
              not        al
              and        al, 80h
              mov        cl, 7
              shr        al, cl
              mov        MpuMode, al
```

```
          pop      dx
          pop      cx
          pop      ax
          ret

SuspendMPU         endp

;-----------------------------------------------------------------------------;
;         SuspendWSS
;
;   ENTRY:
;                  Nothing.
;
;   EXIT:
;                  Nothing.
;
;   USES:
;                  CX, DX = I/O Port address
;-----------------------------------------------------------------------------;
SuspendWSS         proc near

          push     ax
          push     cx
          push     dx

          mov      dx, WssBase
          add      dx, 4                          ; Set dx=WssBase+4

          GetReg   09h, WssInterFace
          and      al, 03h
          jz       SusWSS_Skip1

          xor      al, al
          call     Reg_Write          ; Stop playback & Capture

SusWSS_Skip1:
          inc      dx
          inc      dx                             ; DX = WssBase+6
          in       al, dx
          mov      WssStatus, al
          dec      dx
          dec      dx                             ; DX = WssBase+4

          GetReg   08h, PlayFormat
          GetReg   0Ah, WssControl
          GetReg   0Eh, PlayBaseU
          GetReg   0Fh, PlayBaseL

          xchg     cx, dx
          mov      dx, CtrlBase                   ; CX = WssBase+4, DX = CtrlBase
          GetReg   0Ch, PlayCurrentU
          GetReg   0Bh, PlayCurrentL

          mov      al, WssMode
```

```
                or          al, al
                jz          SusWSS_Skip2

                xchg        cx, dx                          ; CX = CtrlBase, DX = Wssbase+4
                GetReg      1Ch, RecFormat
                GetReg      1Eh, RecBaseU
                GetReg      1Fh, RecBaseL

                xchg        cx, dx                          ; CX = WssBase+4, DX = CtrlBase
                GetReg      0Eh, RecCurrentU
                GetReg      0Dh, RecCurrentL

                xchg        cx, dx                          ; CX = CtrlBase, DX = WssBase+4
                GetReg      10h, DacConfig
                GetReg      15h, WssTimerU
                GetReg      14h, WssTimerL
                GetReg      18h, Istatus

SusWSS_Skip2:
                pop         dx
                pop         cx
                pop         ax
                ret

SuspendWss                  endp

;---------------------------------------------------------------------------;
;               SuspendSbMixer
;
;   ENTRY:
;                           Nothing.
;
;   EXIT:
;                           Nothing.
;
;   USES:
;                           DX = SbBase + 4
;---------------------------------------------------------------------------;
SuspendSbMixer              proc near

                push        ax
                push        dx

                mov         dx, SbBase
                add         dx, 4               ; Set DX = SB Mixer Address Port

                GetReg      04h, SbVoice
                GetReg      0Ah, SbMic
                GetReg      0Ch, SbSource
                GetReg      0Eh, SbSwitch
                GetReg      22h, SbMaster
                GetReg      26h, SbMidi
                GetReg      28h, SbCD
                GetReg      2Eh, SbLine
                pop         dx
                pop         ax
                ret

SuspendSbMixer              endp
```

RESUME.INC

```
;---------------------------------------------------------------------------;
;               SetLogDev
;
;     ENTRY:
;                               Nothing.
;
;     EXIT:
;                               Nothing.
;---------------------------------------------------------------------------;
SetLogDev proc          near

                push            ax
                push            bx
                push            cx
                push            dx

                mov             ah, 02h                         ; Wait for key state
                mov             al, 02h
                call            PnP_Write
                mov             cx, 60h
                call            WaitTime
                call            SendYamahaKey

                mov             ah, 03h
                mov             al, 81h
                call            PnP_Write               ; Write [CSN]

                xor             ah, ah
                mov             al, 80h
                call            PnP_Write               ; Force to set 203h ReadPort

                mov             ah, 06h
                call            PnP_Read

                ; LDN=0 <SA2 Sound System>

                mov             ah, 07h
                mov             al, 00h
                call            PnP_Write               ; Set LDN=0

                ; Write IO port address

                PutRsrcWord 60h, sbBase
                PutRsrcWord 62h, wssBase
                PutRsrcWord 64h, adlibBase
                PutRsrcWord 66h, mpuBase
                PutRsrcWord 68h, ctrlBase
                PutRsrcByte 70h, IRQ_A
                PutRsrcByte 72h, IRQ_B
```

```
        PutRsrcByte 74h, DMA_A
        PutRsrcByte 75h, DMA_B
        PutRsrcByte 30h, Active0

        ; LDN=1 <Joy Stick>

        mov ah, 07h
        mov al, 01h
        call    PnP_Write           ; Set LDN=1

        ; Read IO port address

        PutRsrcWord 60h, JoyBase
        PutRsrcByte 30h, Active1

        mov     ah, 02h
        mov     al, 02h
        call    PnP_Write           ; Wait for key state

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret

SetLogDev endp

;-------------------------------------------------------------------------;
;       WaitForActive
;
;   ENTRY:
;               Nothing.
;
;   EXIT:
;               Carry = 0 : Suceed.
;               Carry = 1 : Error.
;-------------------------------------------------------------------------;
WaitForActive   proc near

        push    ax
        push    cx
        push    dx

        mov     cx, 100
        mov     dx, WssBase
        add     dx, 4

WaitForActive_Loop:
        in      al, dx
        and     al, 80h
        jz      WaitForActive_Exit_Loop
        dec     cx
        jnz     WaitForActive_Loop
```

```
                stc

WaitForActive_Exit_Loop:
                pop             dx
                pop             cx
                pop             ax
                ret

WaitForActive           endp

;------------------------------------------------------------------------;
;               UnmaskDMA
;
;    ENTRY:
;                               Nothing.
;
;    EXIT:
;                               Nothing.
;------------------------------------------------------------------------;
UnmaskDMA               proc near

                push            ax

                mov             al, DMA_A_State
                or              al, al
                jz              UnmaskDMA_Skip
                mov             al, DMA_A
                out             0Ah, al                 ; DMA_A channel unmask

UnmaskDMA_Skip:
                mov             al, DMA_B_State
                or              al, al
                jz              UnmaskDMA_End
                mov             al, DMA_B
                out             0Ah, al                 ; DMA_B channel unmask

UnmaskDMA_End:
                pop             ax
                ret

UnmaskDMA               endp

;------------------------------------------------------------------------;
;               ResumeMixer
;
;    ENTRY:
;                               Nothing.
;
;    EXIT:
;                               Nothing.
;
;    USES:
;                               CX, DX = I/O Port Address
;------------------------------------------------------------------------;
```

```
ResumeMixer         proc near

            push    ax
            push    cx
            push    dx

            mov     al, WssMode
            mov     dx, WssBase
            add     dx, 4
            out     dx, al
            mov     cx, dx                  ; CX = WssBase + 4
            mov     dx, CtrlBase            ; DX = CtrlBase

            mov     ah, 07h
            mov     al, VolumeL
            or      al, 80h                 ; Mute Left Channel
            call    Reg_Write

            mov     ah, 08h
            mov     al, VolumeR
            or      al, 80h                 ; Mute Right Channel
            call    Reg_Write

            xchg    cx, dx                  ; CX = CtrlBase, DX = WssBase+4
            PutReg  00h, InputL
            PutReg  01h, InputR
            PutReg  02h, Aux1L
            PutReg  03h, Aux1R
            PutReg  04h, Aux2L
            PutReg  05h, Aux2R
            PutReg  06h, WaveL
            PutReg  07h, WaveR

            mov     al, WssMode
            or      al, al
            jz      ResumeMixer_Skip

            PutReg  12h, LineL
            PutReg  13h, LineR
            PutReg  1Ah, Mono

ResumeMixer_Skip:
            mov     dx, cx                  ; DX = CtrlBase
            PutReg  09h, Mic
            PutReg  14h, Wide
            PutReg  15h, Bass
            PutReg  16h, Treble
            PutReg  17h, HVInt

            pop     dx
            pop     cx
            pop     ax
            ret
```

```
ResumeMixer         endp

;-----------------------------------------------------------------------;
;           ResumeSBMixer
;
;   ENTRY:
;                   Nothing.
;
;   EXIT:
;                   Nothing.
;
;   USES:
;                   DX = SbBase + 4
;-----------------------------------------------------------------------;
ResumeSBMixer       proc near

            push    ax
            push    dx

            mov     dx, SbBase
            add     dx, 4

            PutReg  04h, SbVoice
            PutReg  0Ah, SbMic
            PutReg  0Ch, SbSource
            PutReg  0Eh, SbSwitch
            PutReg  22h, SbMaster
            PutReg  26h, SbMidi
            PutReg  28h, SbCD
            PutReg  2Eh, SbLine

            pop     dx
            pop     ax
            ret

ResumeSBMixer       endp

;-----------------------------------------------------------------------;
;           ResumeOPL
;
;   ENTRY:
;                   Nothing.
;
;   EXIT:
;                   Nothing.
;
;   USES:
;                   BL = OPL3 Register Array ( 0 or 1 )
;                   DX = Ad lib Base
;-----------------------------------------------------------------------;
ResumeOPL           proc near

            push    ax
            push    bx
            push      dx

            mov       dx, AdlibBase
            mov       bl, 1                           ; Register Array : 1
            PutOPL    05h, Opl3Mode
            xor       bl, bl                          ; Register Array : 0

            PutOPL    08h, Nts
            PutOPL    02h, Opl3Timer1
            PutOPL    03h, Opl3Timer2
            call      WriteOplArray
            PutOPL    0BDh, Rhythm

            mov       al, Opl3Mode
            rcr       al,1
            jnc       ResOpl3_End

            mov       bl, 1                           ; Register Array : 1
            PutOPL    04h, Opl3Connect
            call      WriteOplArray

ResOpl3_End:
            xor       bl, bl                          ; Register Array : 0
            PutOPL    04h, TimerCtrl

            pop       dx
            pop       bx
            pop       ax
            ret

ResumeOPL             endp

;------------------------------------------------------------------------;
;           WriteOPLArray
;
;    ENTRY:
;                     BL = Register Array (0 or 1)
;                     DX = Adlib Base
;
;    EXIT:
;                     Nothing
;
;    USES:
;                     BH = Bit Mask for WriteOPLStatus
;                     CX = Counter
;                     DI = OPL3 Store Address
;                     SI = Address Offset
;
;------------------------------------------------------------------------;
WriteOPLArray         proc near

            push      ax
            push      bx
            push      cx
            push        dx
            push        di
            push        si

            mov         ax, Slot_Size
            mov         cx, ax
            and         bl, 01h
            mul         bl
            mov         si, ax                  ; SI = 0 (BL=0) or Slot_Size (BL=1)

            ; Put Mult

            mov         di, offset Mult0
            add         di, si
            mov         ah, 20h
            call        WriteOplSlot

            ; Put Tl0

            mov         di, offset Tl0
            add         di, si
            mov         ah, 40h
            call        WriteOplSlot

            ; Put Ad

            mov         di, offset Ad0
            add         di, si
            mov         ah, 60h
            call        WriteOplSlot

            ; Put Sr

            mov         di, offset Sr0
            add         di, si
            mov         ah, 80h
            call        WriteOplSlot

            ; Put Ws

            mov         di, offset Ws0
            add         di, si
            mov         ah, 0E0h
            call        WriteOplSlot

            mov         ax, Array_Size
            mov         cx, ax                  ; CX = Counter
            mul         bl
            mov         si, ax                  ; SI = 0 (BL=0) or Slot_Array (BL=1)

            ; Put Fnum

            mov         bh, 0FFh
            mov         di, offset Fnum0
```

```
                add         di, si
                mov         ah, 0A0h
                call        WriteOplStatus

                ; Put Block

                mov         bh, 0DFh
                mov         di, offset Block0
                add         di, si
                mov         ah, 0B0h
                call        WriteOplStatus

                ; Put Fb

                mov         bh, 0FFh
                mov         di, offset Fb0
                add         di, si
                mov         ah, 0C0h
                call        WriteOplStatus

                pop         di
                pop         si
                pop         dx
                pop         cx
                pop         bx
                pop         ax

                ret

WriteOPLArray               endp

;--------------------------------------------------------------------------;
;               WriteOPLSlot
;
;    ENTRY:
;                           AH = Index Base
;                           BL = Register Array ( 0 or 1 )
;                           CX = Counter
;                           DX = Adlib Base Address
;                           DI = Store Address Base
;
;    EXIT:
;                           Nothing.
;
;    USES:
;                           AH = OPL3 Index
;                           BH = Index Base(Stored)
;                           SI = OPL3 Slot Address
;
;--------------------------------------------------------------------------;
WriteOPLSlot                proc near

                push        ax
                push        bx
```

```
            push        cx
            push        si
            push        di

            mov         si, offset Opl3SlotNo
            mov         bh, ah                  ; Store Index base

WriteOplSlot_Loop:
            mov         ah, [si]
            add         ah, bh                  ; ah = Index Base + Index Offset
            mov         al, [di]
            call        PutOplReg
            inc         si
            inc         di
            dec         cx
            jnz         WriteOplSlot_Loop

            pop         di
            pop         si
            pop         cx
            pop         bx
            pop         ax

            ret

WriteOPLSlot            endp

;---------------------------------------------------------------------------;
;           WriteOPLStatus
;
;   ENTRY:
;                       AH = Index Base
;                       BH = Bit Mask
;                       BL = Register Array ( 0 or 1 )
;                       CX = Counter (Index Offset)
;                       DX = Adlib Base
;                       DI = Store Address Base
;
;   EXIT:
;                       Nothing.
;---------------------------------------------------------------------------;
WriteOPLStatus          proc near

            push        ax
            push        cx
            push        di

WriteOplStatus_Loop:
            mov         al, [di]
            and         al, bh
            call        PutOplReg
            inc         ah
            inc         di
            dec         cx
```

```
                jnz             WriteOplStatus_Loop

                pop             di
                pop             cx
                pop             ax
                ret

WriteOPLStatus          endp

;-----------------------------------------------------------------------;
;               PutOPLReg
;
;    ENTRY:
;                               AH = Index
;                               AL = Data
;                               BL = Register Array (0 or 1)
;                               DX = Adlib Base
;
;    EXIT:
;                               Nothing.
;-----------------------------------------------------------------------;
PutOPLReg               proc near

                push            bx
                push            cx
                push            dx

                rcr             bl, 1
                jnc             PutOPLReg_Skip
                inc             dx
                inc             dx              ; If Array = 1, DX = AdlibBase + 2

PutOPLReg_Skip:
                call            Reg_Write

                mov             cx, 1
                call            WaitTime

                pop             dx
                pop             cx
                pop             bx
                ret

PutOPLReg               endp

;-----------------------------------------------------------------------;
;               ResumeSB
;
;    ENTRY:
;                               Nothing.
;
;    EXIT:
;                               Nothing.
;-----------------------------------------------------------------------;
```

```
ResumeSB  proc near

          push      ax
          push      bx
          push      cx
          push      dx

          mov       dx, CtrlBase
          mov       ax, 1009h           ; Index = 10h, Data = 09h
          call      Reg_Write
          mov       bx, offset SbScanData
          mov       cl, 27

ResumeSB_Loop1:
          mov       ah, 11h
          mov       al, [bx]
          call      Reg_Write
          mov       ch, 8

ResumeSB_Loop2:
          mov       ax, 100Bh
          call      Reg_Write
          mov       al, 09h
          call      Reg_Write
          dec       ch
          jnz       ResumeSB_Loop2

          inc       bx
          dec       cl
          jnz       ResumeSB_Loop1

          mov       ah, 11h
          mov       al, [bx]
          call      Reg_Write

          mov       ax, 100Bh
          call      Reg_Write
          mov       al, 09h
          call      Reg_Write
          mov       al, 0Bh
          call      Reg_Write
          mov       al, 09h
          call      Reg_Write

          mov       ax, 1000h
          call      Reg_Write

          pop       dx
          pop       cx
          pop       bx
          pop       ax
          ret

ResumeSB  endp
```

```
;--------------------------------------------------------------------------;
;               ResumeMPU
;
;    ENTRY:
;                           Nothing.
;
;    EXIT:
;                           Carry = 0 : Succeed.
;                           Carry = 1 : Error.
;
;    USES:
;                           CH = Error Status
;                           DX = Mpu Base
;--------------------------------------------------------------------------;
ResumeMPU               proc near

                push        ax
                push        cx
                push        dx

                xor         ch, ch                  ; clear error flag
                mov         al, MpuMode
                or          al, al
                jz          ResumeMPU_Exit

                mov         dx, MpuBase
                inc         dx                      ; DX = MpuBase + 1
                mov         al, 03Fh
                out         dx, al
                mov         cl, 10

ResumeMPU_Loop:
                in          al, dx
                and         al, 80h
                jz          ResumeMPU_ExitLoop
                dec         cl
                jnz         ResumeMPU_Loop
                mov         ch, 1                   ; Set Error flag
                jmp         ResumeMPU_Exit

ResumeMPU_ExitLoop:
                dec         dx                      ; DX = MpuBase
                in          al, dx
                cmp         al, 0FEh
                jz          ResumeMPU_Exit
                mov         ch, 1                   ; set error flag

ResumeMPU_Exit:
                pop         dx
                pop         cx
                pop         ax
                rcr         ch, 1                   ; if CH=1, Set carry flag
                ret
```

```
ResumeMPU           endp

;-----------------------------------------------------------------------------;
;           ResumeWSS
;
;   ENTRY:
;                       Nothing.
;
;   EXIT:
;                       Carry = 0 : Succeed.
;                       Carry = 1 : Error.
;
;   USES:
;                       CX, DX = I/O Port Address
;-----------------------------------------------------------------------------;
ResumeWSS           proc near

            push        ax
            push        cx
            push        dx

            mov         dx, WssBase
            add         dx, 4

            PutReg      0Ch, WssMode
            PutReg      0Ah, WssControl

            mov         al, WssInterface
            and         al, 0FCh
            mov         ah, 49h
            call        Reg_Write

            PutReg      48h, PlayFormat
            mov         cl, 100

ResumeWss_Loop:
            in          al, dx
            and         al, 80h
            jz          ResumeWss_Loop_Exit
            dec         cl
            jnz         ResumeWss_Loop

ResumeWss_Error:
            pop         dx
            pop         cx
            pop         ax
            stc
            ret

ResumeWss_Loop_Exit:
            mov         al, 08h
            out         dx, al
            call        WssWait
            jc          ResumeWss_Error
            PutReg      0Fh, PlayBaseL
            PutReg      0Eh, PlayBaseU

            xchg        cx, dx
            mov         dx, CtrlBase          ; CX = WssBase, DX = CtrlBase
            PutReg      0Bh, PlayCurrentL
            PutReg      0Ch, PlayCurrentU
            xchg        cx, dx                ; CX = CtrlBase, DX = WssBase + 4

            mov         al, WssMode
            or          al, al
            jz          ResumeWss_Skip
            PutReg      05Ch, PlayFormat
            call        WssWait
            jc          ResumeWss_Error

ResumeWss_Skip:
            PutReg      1Fh, RecBaseL
            PutReg      1Eh, RecBaseU

            mov         al, DacConfig
            and         al, 10111111b ; Not 40h
            mov         ah, 10h
            call        Reg_Write

            PutReg      15h, WssTimerU
            PutReg      14h, WssTimerL

            xchg        cx, dx                ; CX = WssBase+4, DX = CtrlBase
            PutReg      0Eh, RecCurrentU
            PutReg      0Dh, RecCurrentL

            mov         al, WssStatus
            or          al, al
            jz          ResumeWss_Skip3

            mov         al, 1
            mov         ah, WssMode
            rcr         ah, 1
            jnc         ResumeWss_Skip2
            mov         al, Istatus
            push        cx
            mov         cl, 4
            shr         al, cl
            pop         cx

ResumeWss_Skip2:
            mov         ah, 0Fh
            call        Reg_Write
            xor         al, al
            call        Reg_Write

ResumeWss_Skip3:
```

```
            xchg        cx, dx                  ; CX = CtrlBase, DX = WssBase + 4
            PutReg      09h, WssInterface
            mov         al, WssMode
            or          al, al
            jz          ResumeWss_End
            PutReg      10h, DacConfig

ResumeWss_End:
            pop         dx
            pop         cx
            pop         ax
            clc
            ret

ResumeWSS               endp

;--------------------------------------------------------------------------;
;           WssWait
;
;   ENTRY:
;                       DX = WssBase + 4
;
;   EXIT:
;                       Carry = 0 : Succeed.
;                       Carry = 1 : Error.
;--------------------------------------------------------------------------;
WssWait     proc near

            push        ax
            push        cx
            push        dx

            mov         al, WssInterface
            and         al, 08h
            jz          WssWait_End

            mov         al, 0Bh
            out         dx, al
            mov         cx, 1000
            call        WaitTime
            inc         dx                      ; DX = WssBase + 5
            mov         cx, 1000

WssWait_Loop:
            in          al, dx
            and         al, 20h
            jz          WssWait_End
            dec         cx
            jnz         WssWait_Loop

            ; ResumeWSS Error
            stc

WssWait_End:
```

```
                pop         dx
                pop         cx
                pop         ax
                ret

WssWait     endp

;--------------------------------------------------------------------------;
;               UnMuteMaster
;
;     ENTRY:
;                           Nothing.
;
;     EXIT:
;                           Nothing.
;--------------------------------------------------------------------------;
UnmuteMaster                proc near

                push        ax
                push        dx

                mov         dx, CtrlBase
                PutReg      07h, VolumeL
                PutReg      08h, VolumeR

                pop         dx
                pop         ax
                ret

UnmuteMaster                endp

;--------------------------------------------------------------------------;
;               ResumeCtrl
;
;     ENTRY:
;                           Nothing.
;
;     EXIT:
;                           Nothing.
;--------------------------------------------------------------------------;
ResumeCtrl                  proc near

                push        ax
                push        dx

                mov         dx, CtrlBase
                PutReg      02h, CtrlIndex
                PutReg      03h, IRQchannel
                PutReg      06h, DMAchannel
                PutReg      0Ah, CtrlMisc

                pop         dx
                pop         ax
                ret
```

```
ResumeCtrl          endp

;-------------------------------------------------------------------------;
;           CurrentIndex
;
;   ENTRY:
;                   Nothing.
;
;   EXIT:
;                   Nothing.
;-------------------------------------------------------------------------;
CurrentIndex        proc near

            push    ax
            push    dx

            mov     dx, WssBase
            add     dx, 4
            in      al, dx
            mov     WssIndex, al

            mov     dx, CtrlBase
            in      al, dx
            mov     CtrlIndex, al

            pop     dx
            pop     ax
            ret

CurrentIndex        endp
```

DATA.INC

```
ResourceData        label byte
          SbBase              dw ?
          WssBase             dw ?
          AdlibBase  dw ?
          MpuBase             dw ?
          CtrlBase   dw ?
          JoyBase             dw ?
          IRQ_A               db ?
          IRQ_B               db ?
          DMA_A               db ?
          DMA_B               db ?
          Active0             db ?
          Active1             db ?

MixerRegister       label byte
          VolumeL             db ?
          VolumeR             db ?
          InputL              db ?
          InputR              db ?
          Aux1L               db ?
          Aux1R               db ?
          Aux2L               db ?
          Aux2R               db ?
          WaveL               db ?
          WaveR               db ?
          LineL               db ?
          LineR               db ?
          Mono                db ?
          Mic                 db ?
          Wide                db ?
          Bass                db ?
          Treble              db ?
          HVInt               db ?

SbMixer             label     byte
          SbVoice             db ?
          SbMic               db ?
          SbSource   db ?
          SbSwitch   db ?
          SbMaster   db ?
          SbMidi              db ?
          SbCD                db ?
          SbLine              db ?

WssRegister         label byte
          WssIndex   db ?
          WssMode             db ?
          WssInterface        db ?
          WssStatus db ?
          PlayFormat          db ?
          WssControl          db ?
          PlayBaseU db ?
          PlayBaseL db ?
          PlayCurrentU        db ?
          PlayCurrentL        db ?
          RecFormat db ?
          RecBaseU  db ?
          RecBaseL  db ?
          RecCurrentU         db ?
          RecCurrentL         db ?
          DacConfig db ?
          WssTimerU           db ?
          WssTimerLdb ?
          Istatus             db ?

PowerManagement     label byte
          Clock               db ?
          Pdn                 db ?
          Psv                 db ?
          Asave               db ?
          Part1               db ?
          Part2               db ?

CtrlRegister        label byte
          CtrlIndex db ?
          CtrlSystem db ?
          IRQchannel          db ?
          DMAchannel          db ?
          CtrlMisc  db ?

Opl3Register        label byte
          Opl3Mode  db ?
          Nts                 db ?
          Opl3Timer1          db ?
          Opl3Timer2          db ?
          TimerCtrl db ?
          Mult0               db Slot_Size dup (?)
          Mult1               db Slot_Size dup (?)
          Tl0                 db Slot_Size dup (?)
          Tl1                 db Slot_Size dup (?)
          Ad0                 db Slot_Size dup (?)
          Ad1                 db Slot_Size dup (?)
          Sr0                 db Slot_Size dup (?)
          Sr1                 db Slot_Size dup (?)
          Ws0                 db Slot_Size dup (?)
          Ws1                 db Slot_Size dup (?)
          Fnum0               db Array_Size dup (?)
          Fnum1               db Array_Size dup (?)
          Block0              db Array_Size dup (?)
          Block1              db Array_Size dup (?)
          Fb0                 db Array_Size dup (?)
          Fb1                 db Array_Size dup (?)
          Rhythm              db ?
          Opl3Connect         db ?

          SbScanData                    db 28 dup(?)
          DMA_A_State                   db ?
          DMA_B_State                   db ?
          MpuMode                       db ?

Opl_Size  EQU       ($-ResourceData)

YamahaKey           label     byte
                    db        0b1h, 0d8h, 06ch, 036h, 09bh, 04dh, 0a6h, 0d3h
                    db        069h, 0b4h, 05ah, 0adh, 0d6h, 0ebh, 075h, 0bah
                    db        0ddh, 0eeh, 0f7h, 07bh, 03dh, 09eh, 0cfh, 067h
                    db        033h, 019h, 08ch, 046h, 0a3h, 051h, 0a8h, 054h
YamahaKey_Size      EQU       ($-YamahaKey)

Opl3SlotNo          LABEL     BYTE
                    db        00h, 01h, 02h, 03h, 04h, 05h
                    db        08h, 09h, 0ah, 0bh, 0ch, 0dh
                    db        10h, 11h, 12h, 13h, 14h, 15h
```

Appendix:

SuspendSB for YMF715E (suspend.c)

```
//----------------------------------------------------------------------
//          Suspend SB routine
//
//          All internal state of SB portion can be read by using scan register.
//          The size of scanned data is 228bit.
//----------------------------------------------------------------------
void        suspendSB(void)
{
 int i,j,t;

 t = 2;

 ctrlWR(0x10,0x01);                        // set SBPDR
 while (1) {
            if ((ctrlRD(0x10) & 0x80) == 0x80) break;
 }

 ctrlWR(0x10,0x0D);                        // ss=1 sm=1 se=0 sbpdr=1
 for (i = 0;i < 28;++i)                    // loop counter is modified for YMF715E
 {
            for (j = 0;j < 8;++j)          // generate 8 clocks
            {
                        ctrlWR(0x10,0x0F); // ss=1 sm=1 se=1
                        ctrlWR(0x10,0x0D); // ss=1 sm=1 se=0
                        wait(t);
            }
            scanData[i] = ctrlRD(0x11);    // read byte in shift register
 }

 for (j = 0;j < 4;++j)                     // generate the last clocks
 {                                         // loop counter is modified for YMF715E
            ctrlWR(0x10,0x0F);             // ss=1 sm=1 se=1
            ctrlWR(0x10,0x0D);             // ss=1 sm=1 se=0
            wait(t);
 }
 scanData[i] = (ctrlRD(0x11) & 0x03) << 6;
 ctrlWR(0x10,0x01);                        // ss=0 sm=0 se=0

}
```

ResumeSB for YMF715E (resume.c)

```
//--------------------------------------------------------------------------
//          Resume SB Routine
//--------------------------------------------------------------------------
void        resumeSB(void)
{
 int i,j,t;

 t = 2;

 ctrlWR(0x10,0x09);                 // ss=1 sm=0 se=0 sbpdr=1
 for(i = 0;i < 28;++i)              // loop counter is modified for YMF715E
 {
            ctrlWR(0x11, scanData[i]);
            for(j = 0;j < 8;++j)    // generate 8 clocks
            {
                        ctrlWR(0x10,0x0B);    // ss=1 sm=0 se=1 sbpdr=1
                        ctrlWR(0x10,0x09);    // ss=1 sm=0 se=0 sbpdr=1
                        wait(t);
            }
 }
 ctrlWR(0x11, scanData[i]);
 for (j = 0;j < 4;++j)              // generate the last clocks
 {                                  // loop counter is modified for YMF715E
            ctrlWR(0x10,0x0B);      // ss=1 sm=0 se=1 sbpdr=1
            ctrlWR(0x10,0x09);      // ss=1 sm=0 se=0 sbpdr=1
            wait(t);
 }

 ctrlWR(0x10,0x00);                  // ss=0 sm=0 se=0 sbpdr=0
}
```

SuspendSB for YMF715E (suspend.inc)

```
;---------------------------------------------------------------------------;
;               SuspendSB
;
;     ENTRY:
;                               Nothing
;
;     EXIT:
;                               Carry = 0 : Suceeded.
;                               Carry = 1 : Error.
;---------------------------------------------------------------------------;
SuspendSB proc near

                push        ax
                push        bx
                push        cx
                push        dx

                mov         dx, CtrlBase
                mov         ax, 1001h               ; Index=10h, Data=01h
                call        Reg_Write               ; Set SBPDR
                mov         cx, 100h

SusSB_Loop1:
                call        Reg_Read
                and         al, 80h
                jnz         SusSB_Loop1Exit
                dec         cx
                jnz         SusSB_Loop1

                ; SusSB_Error

                pop         dx
                pop         cx
                pop         bx
                pop         ax
                stc
                ret

SusSB_Loop1Exit:
                mov         ax, 100Dh  ; Index=10h, Data=0Dh
                call        Reg_Write
                mov         bx, offset SbScanData
                mov         cl, 28         ; loop counter is modified

SusSB_Loop2:
                mov         ch, 8

SusSB_Loop3:
                mov         ax, 100Fh
                call        Reg_Write
                mov         al, 0Dh
                call        Reg_Write
```

```
            dec     ch
            jnz     SusSB_Loop3

            mov     ah, 11h
            call    Reg_Read
            mov     [bx], al
            inc     bx
            dec     cl
            jnz     SusSB_Loop2

            mov     ax, 100Fh
            call    Reg_Write
            mov     al, 0Dh
            call    Reg_Write
            mov     al, 0Fh
            call    Reg_Write
            mov     al, 0Dh
            call    Reg_Write

            ; add more 2 pulse for YMF715E

            mov     al, 0Fh
            call    Reg_Write
            mov     al, 0Dh
            call    Reg_Write
            mov     al, 0Fh
            call    Reg_Write
            mov     al, 0Dh
            call    Reg_Write

            mov     ah, 11h
            call    Reg_Read
            mov     cl, 6
            sal     al, cl
            mov     [bx], al

            mov     ax, 1001h   ; Index =10h, Data = 01h
            call    Reg_Write

            pop     dx
            pop     cx
            pop     bx
            pop     ax
            clc
            ret

SuspendSB endp
```

ResumeSB for YMF715E (resume.inc)

```
;-----------------------------------------------------------------------;
;           ResumeSB
;
;   ENTRY:
;                   Nothing.
;
;   EXIT:
;                   Nothing.
;-----------------------------------------------------------------------;
ResumeSB  proc near

            push    ax
            push    bx
            push    cx
            push    dx

            mov     dx, CtrlBase
            mov     ax, 1009h           ; Index = 10h, Data = 09h
            call    Reg_Write
            mov     bx, offset SbScanData
            mov     cl, 28              ; loop counter is modified for YMF715E

ResumeSB_Loop1:
            mov     ah, 11h
            mov     al, [bx]
            call    Reg_Write
            mov     ch, 8

ResumeSB_Loop2:
            mov     ax, 100Bh
            call    Reg_Write
            mov     al, 09h
            call    Reg_Write
            dec     ch
            jnz     ResumeSB_Loop2

            inc     bx
            dec     cl
            jnz     ResumeSB_Loop1

            mov     ah, 11h
            mov     al, [bx]
            call    Reg_Write

            mov     ax, 100Bh
            call    Reg_Write
            mov     al, 09h
            call    Reg_Write
            mov     al, 0Bh
            call    Reg_Write
            mov     al, 09h
            call    Reg_Write
```

```
        ; add more 2 pulse for YMF715E

        mov     al, 0Bh
        call    Reg_Write
        mov     al, 09h
        call    Reg_Write
        mov     al, 0Bh
        call    Reg_Write
        mov     al, 09h
        call    Reg_Write

        mov     ax, 1000h
        call    Reg_Write

        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret

ResumeSB endp
```

Release Note :

Preliminary Ver0.9
March 18th 1997

Ver1.00
May 8th 1997
Correct some miss coding.
Add SB mixer register.
Add OPL4ML(2) description

Ver1.01
May 9th 1997
Correct some miss coding.

Ver1.02
May 9th 1997
Correct some miss coding. (opl connect address)

Ver1.10
July 10th 1997
Modified some Description and coding.
Add Sample Assembler Code for Suspend/Resume
Put Mask DMA to the first procedure of suspend.

Ver1.20
August 1st 1997
Modified scandata array size in suspend.c/resume.c
Add Sample Assembler and C Code for YMF715E