1052 Elwell Court, Palo Alto, CA 94303 Tel: 800.683.5943 www.zfmicro.com

Booting User Software from Flash

Often, HW/SW developers design solutions using a single Flash device as the source of the BIOS, OS and/or application. This document focuses on booting Linux from a single Flash chip on the ZFx86 Integrated Development System (IDS). Apply this theory to booting other operating systems such as WindRiver's VxWorks™ or other user-written special application programs.

This BIOS-independent approach utilizes ZFx86 specific features built into the latest ZFx86 Phoenix BIOS. Download the latest Phoenix BIOS from the ZF Micro Devices website:
http://www.zfmicro.com

Using Option-ROMs

ZFx86's Flash-software booting approach relies on the option-ROM scan system, a feature found in all AT-compatible PCs. The common ISA video card BIOS is considered an option-ROM; although, it is a special case which gets executed in the very early stage of the BIOS Power On Self Test (POST) sequence. Other option-ROM examples are: ROM-BASIC used on early ATs, and all firmware on PCI or ISA extension boards (for example, network interface controllers, or SCSI controllers).

During the POST sequence, the BIOS performs a so-called ROM-scan sequence:

- The BIOS looks at the beginning of every 2kbyte block in the address region C8000h through F0000h to find a "55 AA" signature.
- When it finds the "55 AA" signature, the next byte determines the option-ROM size in 512-byte increments.
- For detecting corruption, the sum of all option-ROM bytes must equal 100h. Usually achieved by setting the last byte to: 100h minus the sum of all other option-ROM bytes.
- When the BIOS validates the option-ROM is correct, it calls the routine starting at offset 03h.

The option-ROM routine completes various tasks; typically, it initializes the hardware to some known state and hooks some interrupt vectors used later by other OS services or user programs. ZFx86 uses the option-ROM code to boot Linux from Flash. We call this routine the Linux Loader (LL).

This process works for all possible HW configurations where the option-ROM *and* the BIOS may reside in the same device or in different chips on the motherboard. You must correctly set or route the ZFx86's Chip Select signals and the custom BIOS settings to the chip region where the option-ROM resides, and address the option-ROM during the POST sequence.

See the ZF Micro Device's website for compressed LinuxLoaderOptionROM.zip and VxWorksOptionROM.zip files.

Booting Initalization Sequence

The following identifies the startup sequence the loader completes:

Initialize the CPU and cache

Initialize Lambda North Brigde

Initialize the South Bridge

Program the SuperIO

Size and initialize the system memory

Copy the loader to shadow RAM

Set up the stack and interrupt vectors

Initialize the Serial Port (9600 baud, 8 bit, no parity, handshake set to none)

Perform PCI bus scan and allocate resources for the PCI devices

Set up interrupt controllers and system timer chip

Initialize system Real Time Clock

Set up Video BIOS if present (ISA bus card given priority over PCI)

Output the amount of memory detected in system and the PCI device list

Open Flash window with following parameters:

Start address in Flash = 1B0000h

System Base Address = D0000h

Size = 2000h

Perform Option Rom scan from D0000h to EFFFFh

Execute Option Rom image if found, otherwise bring the system to a halt.

The Linux Loader Operation

The Linux Loader (LL) copies the Linux kernel from a Flash address to a RAM address that matches the address used when Linux starts from the Hard Disk.

Also, depending on the default compiled options, the kernel identifies whether to mount its root file system from a RAM disk or from a hard disk partition. Initally, the Linux Loader stores the RAM disk root file system in Flash as a compressed "*initrd*" image (initialize RAM-disk) and copies it to the end of the available RAM. The *initrd* image begins with a 4-byte header value that indicates how many bytes to copy from Flash-to-RAM.

Download the "*Booting Linux from Flash*" Application Note (P/N 9150-0017-00) for detailed instructions on using the Linux Loader, the ZFlash Linux Loader.zip file containing a sample *initrd* image and a Linux File System. See the ZF Micro Devices website: *http://www.zfmicro.com.*

The following procedure describes the internal working of the Linux Loader. Use these same concepts to launch other operating systems.

1. Invoke the Linux Loader using the special option-ROM scan routine, which gains control just before normal boot process and scans the memory window mem_cs0 settings defined in PhoenixBIOS Setup Utility > Advanced > Advanced Chipset Control > ISA Memory Chip Select Setup menu. See Figure 1.

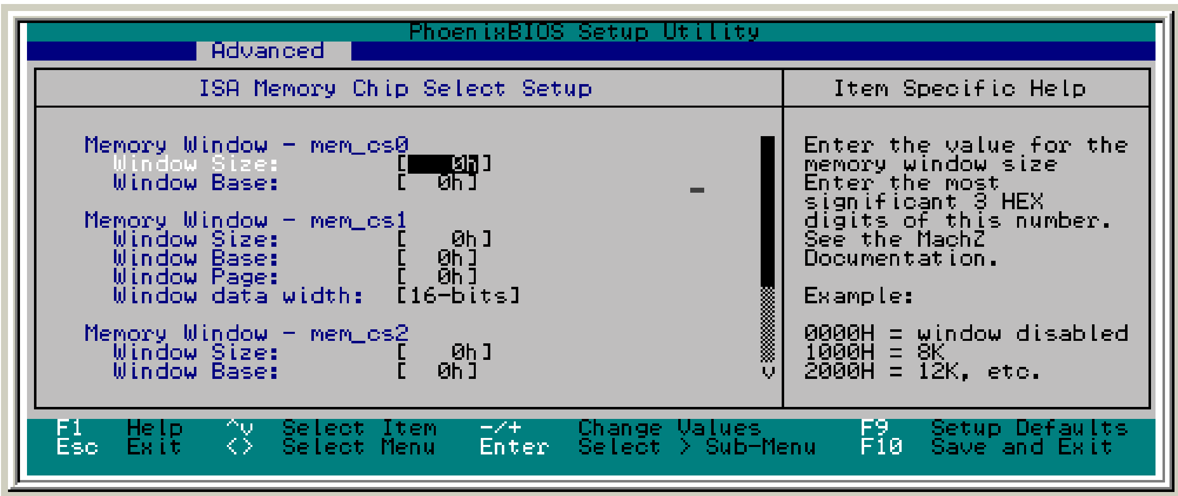

Figure 1. ISA Memory Chip Select Setup Menu

Use the memory window settings to map a specific region from Flash memory to the desired address below the 1Mb boundary (by default, the chip select setting maps part of the BIOS from the end of 2Mb Flash device to address E0000h).

Set the mem_cs0 to any needed value, because at the time point when those settings are required, the BIOS is already shadowed and nolonger executing from the Flash device.

2. As a first step after execution, the LL relocates itself to the main working memory address 9B00:0000. Because in a later processing step, the LL redefines the mem_cs0 memory window, and if at this point the code is still working from the Flash device (that is, mapped to a memory window below 1Mb), the original mem_cs0 value disappears from the initial memory window and a total system crash occurs.
3. The LL initializes the Serial Port to allow for diagnostic messages.
4. Then, the LL waits 3 seconds to allow for user input.
   - If the ESC key is pressed, the loader quits, and the BIOS' special option-ROM scan routine regains control.
   - If the option-ROM scan routine cannot find any other valid option-ROMs in the defined search area (defined by mem_cs0 settings), a normal disk boot occurs.
5. After the 3 seconds elapses without the ESC key pressed, the loading sequence starts.

   a. The kernel saves the original mem_cs0 Flash window settings and enlarges the Flash window to 16MB using the ZFx86's chip select programming features. This Flash window is visible for all available memory addresses which are not claimed by the memory controller (that is, addresses reserved for RAM) or PCI devices due to the fact that the ISA bus has a lower priority than all the other chip devices (for example, memory controller or PCI controller).

b. Therefore, at the end of the DRAM memory map, the mapped Flash content is visable over the entire upper memory space at every 16MByte boundary.

- For instance, the LL repeats the entire 16MByte Flash contents in the address range 10000000h through 10FFFFFFh (also at 11000000h through 11FFFFFFh, 12000000h through 12FFFFFFh, and so on).

Note: To access the entire upper memory space, enable the A20M# line.

6. Before the LL copies large amounts of data from the extended memory to the lower RAM, it must account for the processor's protected mode operation.

   a. First the LL initializes the Global Descriptor Table (GDT) so that the data segment size increases from the normal 64KB to 4GB.

   b. Then, it loads this GDT data.

   c. Switchs the processor to protected mode.

   d. Sets data segment DS and extra segment ES selectors with the previously defined 4GB data range GDT entries.

   e. Then switchs the processor back to real mode again.

   This allows you to access the entire 4GB memory space in real mode as long as the DS and ES registers are not overwritten.

7. Once the LL access the full memory space, it checks for the Linux kernel's signature at Flash offset 202h (in our example, visible at memory address 10000202h).

   - If the signature is not found, the LL increments the search address with a 16MB value and checks again for a signature.
   - If the kernel signature is not found after 10 checks, the LL restores the original memory window settings, prints out the "*Linux kernel setup signature not found"* message and returns.

   Note: The LL searches the signature addresses at 10000202h,11000202h, 12000202h, and so on,

8. The kernel header data structure contains bootstrap code, and kernel setup code. The LL copies this code from the previous address (in our example, 10000000h) to 90000h in low memory.

9. The LL reads the kernel's loading address from the header and then copies the kernel itself to the correct loading address in RAM.

   - In a normal sized kernel, it loads at address 10000h.
   - In a large sized kernel (made using "make bzImage"), it loads at the high memory address 100000h.

10. In order to load the *initrd* image to RAM, the LL requests the detected "top of memory"

system address from the ZFx86 South Bridge. The Linux Loader requests that the *initrd* image start at Flash offset 80000h.

a. Before the *initrd* gets copied, the LL checks for its presence by reading the *initrd* size value from Flash offset 80000h (memory address 10080000h, 11080000h, and so on).

b. If the size value is 0 or 0FFFFFFFFh, the LL skips copying the *initrd*; otherwise, it copies the *initrd* image to the end of the detected RAM without the 4-byte length header.

11. The LL writes the *initrd* size and start address to the kernel setup parameter block which resides in memory location starting at 90200h. If *initrd* was not found, the LL zeros out these values.
12. The LL now closes the previously created 16MB Flash window and restores the original mem_cs0 window.
13. To boot the kernel normally, the LL updates the boot sector data area at 90000h with the values needed to configure the kernel. For example, those values might be that there are 4 setup sectors, that the root device is read-only, that the ram disk is 0, that the swap size is 0, and so on, also that the system size, the video mode, and the root device name are set, and that the stack is set up to the kernel setup code's stack area.
    - The root device name is based on the compiled-in root-device id value, where 100h=ram0, 301h=hda1, 302h=hda2, 303h=hda3, 304h=hda4, 0=disabled.
    - If the root device id is set to 100h, the root device will be the one contained in compressed form in our *initrd* image in Flash.
    - If root device id is set to 301h, then the loader mounts the root device from the first hard disk partition or /dev/hda1.
14. The final action to start Linux is a Far Jump to the beginning of the kernel setup code.

    The kernel boot messages begin appearing on the screen or COM-port, and the root file system mounts from RAM-disk or the hard disk partition depending on the compiled-in Root_Device variable.
15. The Linux login prompt displays.

Linux Loader Flowchart

Figure 2 charts the Linux Loader's logic flow.

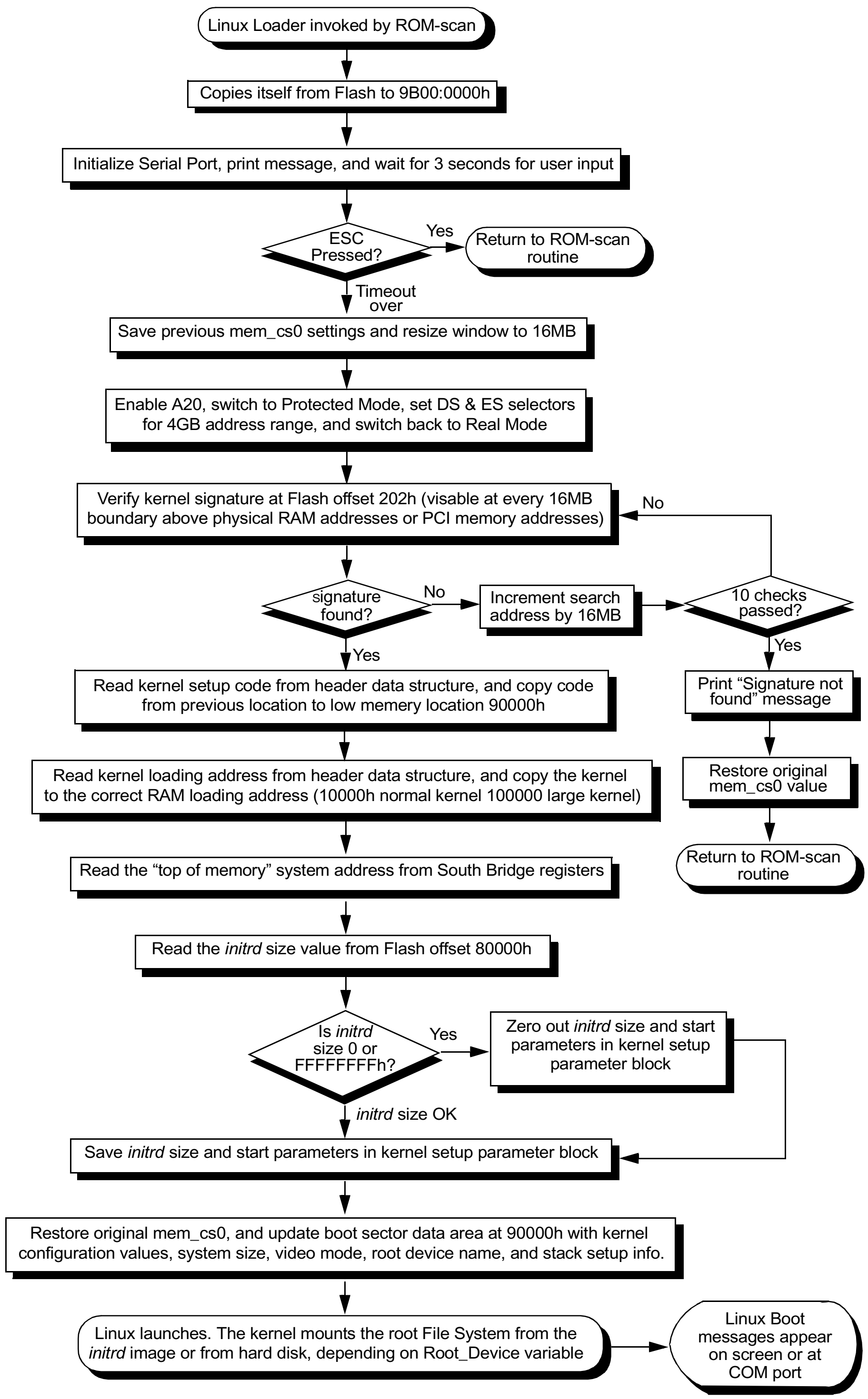

Figure 2. The Linux Loader Flow Chart

The Flash layout

The Flash layout depends on the actual system hardware set up and the amout of available Flash memory. The following items are required:

- System BIOS or special initialization code (ZFx86 Phoenix BIOS) which supports option-ROMs
- The Linux Loader image converted to option-ROM format with the needed headers and checksums included
- The Linux kernel – exactly the same file created as the end product of the kernel compilation
- Optional *initrd* image (a compressed root File System image that expands as a RAM-disk). You may omit this otional image if the root File System resides on an alternate device (for example, IDE, Compact Flash, Disk on Chip, and so on) and mounts from the alternate device.

For example, your design may contain a large Flash chip such as the 16MByte Intel E28F128 StrataFlash. In this case, organize the Flash memory layout as follows:

Start offset	**Item**
FC0000	Phoenix BIOS 256K
FB8000	Linux Loader
080000	*initrd* image beginning with 4-byte header, up to address FB7FFF
000000	Linux Kernel, up to address 07FFFFh

Map the Linux Loader to D8000 using the BIOS' internal memory chip select mapping mechanism. The ZFx86 Phoenix BIOS allows the creation of up to four memory windows using chip selects mem_cs0 through mem_cs3. Although, the BIOS uses the mem_cs0 to start from Flash after reset; the BIOS is then shadowed into RAM thereby allowing us to reprogram mem_cs0 for other memory windows. During the special option-ROM scan, the Linux Loader maps to the correct location in RAM, locates it, and executes.

In order to map the Linux Loader to mem_cs0, select the desired place in RAM using the Phoenix BIOS Setup Utility > Advanced > Advanced chipset Control > ISA Memory Chip Select Setup menu and set the following values:

Window Size:	1 – sets window size to 8kb
Window Base:	D8 – sets window base to D8000h
Window Page:	EE0 – sets Flash page register value to 1000000h–D8000h+FB8000h = EE0000h
Window Data Width:	16 or 8 based upon the data path width of the device used in your design.

Using the Z-tag Manager

Figure 3 shows the Z-tag Manager Contents window.

Z-tag Contents - 18 items

Id	Name	Ver	CRC	Date	Time	Body_len
02	Select Serial Device	0001	1021	20000609	2025	1
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	Kernel Image at 000000	0001	0000	20000724	1614	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	Kernel Image	0001	0000	20000925	1335	340818
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	Initrd Image start	0001	89A9	20000724	1626	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	toytest16 initrd	0001	1F2F	20000927	1031	1736134
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	Linux Loader at FB8000	0001	0000	20001121	1252	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	Linux Loader as ROM ext	0001	0000	20001201	1654	1536
01	Strataflash Programmer	0001	C27A	20001011	1815	3435
FE	BIOS start FC0000	0001	56AC	20001121	1252	4
FE	Erase Sector =1	0001	0000	20000724	1614	4
FF	Phoenix A10 BIOS Image	0001	0000	20001122	1140	262144
05	Stop Processing	0001	0000	20001121	1253	0

Figure 3. Z-tag Manager's Contents Window Defining Data To Be Flashed

For more detailed instructions, see "*Booting Linux From Flash*" (P/N 9150-0017-00) document on the ZF Micro Device website: *http://www.zfmicro.com*

1. Use the Z-tag Manager configured for Pass Through mode to load the data.
2. Connect the parallel port extension cable to your development host computer.
3. Connect the Z-tag dongle (JP2 pins 2-3 jumpered for PassThrough mode) to the extension cable and to your target board's Z-tag connector.
4. Verify that Chip Select 0 is jumpered to your selected target Flash chip.
5. Press the Z-tag Manager's Write-button and reset the target board to initiate the download and Flash burning sequence.

   The Z-tag Manager operation is documented in the Z-tag Manager manual and in other reference documents from ZFMicro Devices.
6. To monitor the download progress, connect the serial cable from the target board's COM1 port to your development host computer's COM port. Set the COM port to the following:
   - Speed 9600 baud
   - 8 bit, no parity
   - Handshake set to none

Conclusion

Loading and launching Linux from Flash is not a complicated task if a Linux Loader binary is contained in an option-ROM using a compatible format. You might need several images for different purposes, for example, for mounting root file system from RAM disk, or for mounting the root file system from /dev/hda1.

- Use the BIOS or other system initialization code to set up the hardware properly and detect the amount of memory installed in your system.
- The BIOS or system initialization code then launchs the Linux Loader (or some other operating system loader which you build using the same general principles) either during the option-ROM scan or by a direct jump to it.
- In case of a complete Linux system, place both the kernel at offset 0h and the *initrd* images at offset 80000h in the Flash.
- The *initrd* image must contain a 4 byte-long image-length header before the actual image starts.
- For mounting the root file system from a hard disk, you only need the kernel in the Flash. Compile the Linux Loader with correctly defined Root_Device id settings.
- Generally, you can modify the current Linux Loader code to match your HW design, and the images may reside in completely different Flash offsets.

Appendix A: The Linux Loader Source Code

```
; ORLL (Option-ROM Linux Loader) v1.00
; Last modified on 18.01.2001

       .model    tiny
       .486p

; Linux root device options:
; 100h=ram0, 301h=hda1, 302h=hda2, 303h=hda3, 304h=hda4, 0=disabled

Root_Deviceequ   301h
Serial_Addrequ   03f8h               ; 3F8h = COM1, 2F8h = COM2
Screen_Outputequ 1                   ; 1 = Output messages also to the screen

MSG    MACRO     text
       mov       si,offset text
       call      Output
       ENDM

PCODE  MACRO     postcode
       mov       al,postcode
       out       80h,al
       ENDM

ZFLWB  MACRO     register,value8
       mov       al,register
       mov       dx,218h
       out       dx,al
       inc       dx
       mov       al,value8
       out       dx,al
       ENDM

ZFLRB  MACRO     register
       mov       al,register
       mov       dx,218h
       out       dx,al
       inc       dx
       in        al,dx
       ENDM

ZFLWDW MACRO     register,value32
       mov       al,register
       mov       dx,218h
       out       dx,al
       inc       dx
       inc       dx
       mov       eax,value32
       out       dx,eax
       ENDM

ZFLRDW MACRO     register
       mov       al,register
       mov       dx,218h
       out       dx,al
       inc       dx
```

```
        inc         dx
        in          eax,dx
        ENDM

        .code
        org         0

Start:

        db          55h,0aah              ; Extension ROM signature,
        db          3                     ; and length in 512-byte pages

        PCODE       070h
        mov         ax,cs
        mov         ds,ax
        jmp         Skip_GdtArea

; Global Descriptor Table

        org         10h                   ; For proper alignment

Gdt     dd        0,0                     ; 1st entry, not used
GdtProtdw           0ffffh,0000h          ; 2nd entry
        db          0,93h,8fh,0
GdtDescdw           $-Gdt                 ; GDT size
GdtBasedd           0                     ; GDT base address

Skip_GdtArea:

; First we move our code out of extension ROM space, so we can open new
; 16Mb wide memory window for strataflash where we locate the kernel and
; initrd images. Since this overrides memory window settings of our extension
; ROM, we need to get out of here.
; Bootsector & linux kernel setup goes to 9000:0000, length 0A00h bytes,
; linux kernel itself goes to 1000:0000, max length 08000h bytes,
; which leaves safe location for us below 1000:0000 or above 9000:0A00,
; so I chose 9200:0000. We don't have to worry about this if we have big
; kernel which goes above 1Mb.

New_Segequ          9200h

        mov         ax,New_Seg
        mov         es,ax
        lea         si,Start
        mov         di,si
        mov         cx,offset Loader_End-Start
        cld
        rep         movsb

        db          0eah                            ; Far jump to Start
        dw          offset Loader_Start,New_Seg

Loader_Start:

        PCODE       71h


; Initialize serial port
```

```
        mov     dx,Serial_Addr+3
        mov     al,80h
        out     dx,al                   ; Set DLAB
        mov     dx,Serial_Addr
        mov     ax,12                   ; 12 = 9600 bps
        out     dx,ax                   ; Baud rate divisor
        mov     dx,Serial_Addr+3
        mov     al,3                    ; 3 = 8N1
        out     dx,al                   ; Line mode (8N1)
        mov     dx,Serial_Addr+4
        xor     al,al
        out     dx,al                   ; Clear DTR & RTS
        MSG     T_Loader_Start          ; Output loader startup message

; Here we wait 3 seconds for user input, if ESC key is pressed, loader quits
; with jump to the original Int 19h vector

        mov     ax,40h
        mov     es,ax
        mov     ebx,es:[6ch]
        add     ebx,55          ; We wait 55 timer ticks, ca 3 seconds
@@:
        cmp     ebx,es:[6ch]
        jl      @f
        in      al,60h          ; Check if ESC key has been pressed
        cmp     al,0
        jz      @b
        cmp     al,1
        jne     @f              ; No, go check again have 3 seconds passed yet
        MSG     T_Cancel

; Exit loader

        retf
@@:
        MSG     T_Start

        PCODE   73h

; Save current memory window settings

        lea     edi,offset MemWinISA24
        ZFLRB   5Bh             ; ISA 24-bit address calculation
        stosb
        ZFLRDW  26h             ; Window base
        stosd
        ZFLRDW  2Ah             ; Window size
        stosd
        ZFLRDW  2Eh             ; Window page
        stosd

; Define 16Mb wide memory window for chip select 0

        ZFLWB   5Bh,1           ; Set ISA 24-bit address calculation
        ZFLWDW  26h,0           ; Set base address (actual ports 27h and 28h)
        ZFLWDW  2Ah,1000000h-1  ; Window size is 16MB (strataflash)
        ZFLWDW  2Eh,0           ; Page address
```

```
; Enable A20 line

        PCODE      74h

         cli
        in         al,92h
         jmp$+2
         jmp$+2
        or         al,2                ; Enable A20 bit
        out        92h,al

; Initialize and load GDT

        PCODE      75h

        sub        eax,eax
        mov        ax,cs
        shl        eax,4
        add        eax,offset Gdt
        mov        cs:GdtBase,eax
        lgdt       fword ptr cs:GdtDesc

; Switch processor to protected mode

        mov        eax,cr0
        mov        ebx,eax
        or         ax,1                ; Set PE bit
        mov        cr0,eax             ; Enable protected mode
        jmp        $+2                 ; Flush instruction cache
        mov        ax,(GdtProt-Gdt)
        mov        ds,ax               ; Define selectors for DS

; Switch processor back to real mode

        mov        cr0,ebx             ; Clear PE bit, back to real mode
        jmp        $+2                 ; Flush instruction cache

; Check for Linux kernel setup signature. If we cant find the signature in
; first try, we perform a scan loop on higher addresses just to be sure that
; the address space where we were looking was not claimed by any other device
; with higher priority. This scanning technique is possible because of ISA bus
; being only 24 bits wide and its 16Mb address space gets repeated after every
; 16Mb block through entire 4GB adress space. We start from 10000000h, thats
; above 256Mb, maximum amount of RAM that ZFx86 can be configured with.

        PCODE      76h

        mov        esi,10000000h       ; Start address
        mov        cx,10               ; Number of cycles
@@:
        add        esi,202h            ; Start address + kernel setup signature offset
        mov        eax,[esi]
        cmp        eax,053726448h      ; Look for 'HdrS'
        je         SigFound
        add        esi,01000000h       ; Add 16Mb to the address and try again
        loop       @b
        jmp        NoSignature         ; No signature found, skip the whole thing
```

```
; Now copy kernel setup and bootstrap code

SigFound:

        sub       esi,202h
        mov       ebp,esi              ; EBP = kernel setup start address

        PCODE     77h

        add       si,1f1h
        xor       ax,ax
        mov       al,[esi]             ; Get setup sector size
        inc       al                   ; Add bootstrap sector
        shl       ax,9                 ; Multiply by 512 for size in bytes
        mov       bx,ax                ; Store value for kernel start address
        sub       esi,1f1h             ; Start address of the kernel setup code
        mov       edi,90000h           ; Destination address
        mov       cx,ax
@@:
        mov       eax,ds:[esi]
        mov       ds:[edi],eax
        add       esi,4
        add       edi,4
        sub       cx,4
        jnz       @b

; Now copy kernel image

        PCODE     78h

        mov       esi,ebp
        add       esi,211h
        mov       al,[esi]             ; Kernel boot option
        add       esi,3
        mov       edi,[esi]            ; Kernel load offset in system memory
        sub       esi,214h
        mov       si,bx                ; Start address of the kernel image
        mov       ecx,080000h          ; Maximum kernel size to copy
        or        al,al
        jnz       @f
        shl       edi,4                ; Start address of kernel (10000h or 100000h)
@@:
        mov       eax,[esi]
        mov       [edi],eax
        add       esi,4
        add       edi,4
        sub       ecx,4
        jnz       @b

        PCODE     79h

; Read top of system memory address from south bridge
; This is specific to the ZFx86 BIOS'es

        mov       eax,8000904ch        ; PCI south-bridge top of system memory register
        mov       dx,0cf8h
        out       dx,eax
```

```
        mov     dx,0cfch
        in      eax,dx
        and     al,0f0h                 ; We have to clear lower 4 bits (SB speciality)
        dec     eax

; Check for initrd size/presence in flash rom,
; and copy it to the top of system memory

        PCODE   7Ah

        mov     edi,eax
        mov     esi,ebp
        add     esi,80000h              ; Start address of the initrd image in memory
                                        ; window
        mov     ecx,[esi]               ; Get size of the initrd image
        cmp     ecx,0                   ; Skip initrd if size is zero, means it's
disabled
        jz      SkipInitrd
        cmp     ecx,-1                  ; Also skip initrd if the memory is pobably
        jz      SkipInitrd              ; not initialized

        PCODE   7Bh

        add     esi,4                   ; Skip first 4 bytes of image (initrd size)
        neg     ecx
        add     edi,ecx                 ; Calculate start address of the image in system
memory
        neg     ecx
        xor     di,di
        mov     ebx,edi                 ; Save start address of the image
@@:
        mov     eax,[esi]
        mov     [edi],eax
        add     esi,4
        add     edi,4
        sub     ecx,4
        jc      @f
        jnz     @b
@@:
        mov     esi,ebp
        add     esi,80000h              ; Start address of the initrd image in
                                        ; memory window
        mov     ecx,[esi]               ; Get size of the initrd image
        jmp     @f

SkipInitrd:

        PCODE   7Ch

        sub     ebx,ebx                 ; Set zeroes if initrd was not found in flash
        sub     ecx,ecx

; Write start address and size of ramdisk image (initrd) to the kernel setup
; parameters block

@@:
        PCODE   7Dh
```

```
        mov     ax,9020h                ; Kernel setup segment
        mov     ds,ax
        mov     ds:[24],ebx             ; Start address of initrd image
        mov     ds:[28],ecx             ; Initrd image size

; Restore original memory window

        ZFLWB   5Bh,0                   ; Clear full 24-bit ISA addressing
        ZFLWDW  2Eh,0F00000h            ; Set page
        ZFLWDW  2Ah,10000h-1            ; Window size is 64k
        ZFLWDW  26h,0F0000h             ; Set base address

; Setup parameters

        mov     ax,9000h                ; Bootsector data area
        mov     ds,ax
        mov     byte ptr ds:[1f1h],4    ; Setup sectors
        mov     word ptr ds:[1f2h],1    ; Root flags (read only)
        mov     word ptr ds:[1f4h],8000h ; System size
        mov     word ptr ds:[1f6h],0    ; Swap device
        mov     word ptr ds:[1f8h],0    ; Ramdisk
        mov     word ptr ds:[1fah],0f00h ; VGA screen mode
        mov     word ptr ds:[1fch],Root_Device; Root file system device

; Command line patch

        mov     ds:[020h],0a33fh
        mov     ds:[022h],8cc1h

; Root device name

        cld
        mov     si,offset T_Root_Device
        mov     di,08cc1h
        mov     ax,ds
        mov     es,ax           ; ES=9000h
        mov     ax,cs
        mov     ds,ax           ; DS=CS
@@:
        lodsb
        stosb
        or      al,al
        jne     @b

        mov     ax,9020h                ; Kernel setup segment
        mov     ds,ax

        mov     byte ptr ds:[16],61h    ; Set loader type and version

; Everything is done, now lets jump into kernel setup code

        PCODE   7Eh

        db      0eah                    ; Far jump into kernel setup code
        dw      0,09020h
```

```
NoSignature:

        PCODE     7Fh
        MSG       T_SigNotFound

; Restore original memory window

        lea       si,offset MemWinISA24
        lodsb
        ZFLWB     5Bh,al              ; Clear full ISA addressing bit
        lodsd
        ZFLWDW    2Eh,eax             ; Window page
        lodsd
        ZFLWDW    2Ah,eax             ; Window size
        lodsd
        ZFLWDW    26h,eax             ; Window base
        retf

;-----------------------------------------------------------
;
; Output string from CS:SI ending with zero
;

Output:
        mov       dx,Serial_Addr+5
@@:
        in        al,dx
        test      al,20h
        jz        @b
        mov       al,byte ptr cs:[si]
        inc       si
        cmp       al,0
        jne       @f
        ret
@@:
        sub       dx,5
        out       dx,al
IF Screen_Output EQ 1
        mov       ah,0eh
        int       10h
ENDIF
        jmp       short Output

MemWinISA24db     0
MemWinBasedd      0
MemWinSizedd      0
MemWinPagedd      0

IF Root_Device EQ 100h
T_Root_Devicedb  '/dev/ram0 ',0
ENDIF
IF Root_Device EQ 301h
T_Root_Devicedb  '/dev/hda1 ',0
ENDIF
IF Root_Device EQ 302h
T_Root_Devicedb  '/dev/hda2 ',0
ENDIF
```

```
IF Root_Device EQ 303h
T_Root_Devicedb  '/dev/hda3 ',0
ENDIF
IF Root_Device EQ 304h
T_Root_Devicedb  '/dev/hda4 ',0
ENDIF
IF Root_Device EQ 0
T_Root_Devicedb  0
ENDIF

T_Loader_Startdb 13,10,'Option-ROM Linux loader 1.00, press ESC to cancel...',0
T_Start          db         'starting.',13,10,10,0
T_Canceldb       'cancel.',13,10,0
T_SigNotFounddb  'Linux kernel setup signature not found.',13,10,10,0

Loader_End:
                 end        Start
```