부트로더 구조 및 원리 파악:
Blob 소스 분석

단국대학교
컴퓨터학과
2009
백승재
ibanez1383@dankook.ac.kr
http://embedded.dankook.ac.kr/~ibanez1383

강의 목표

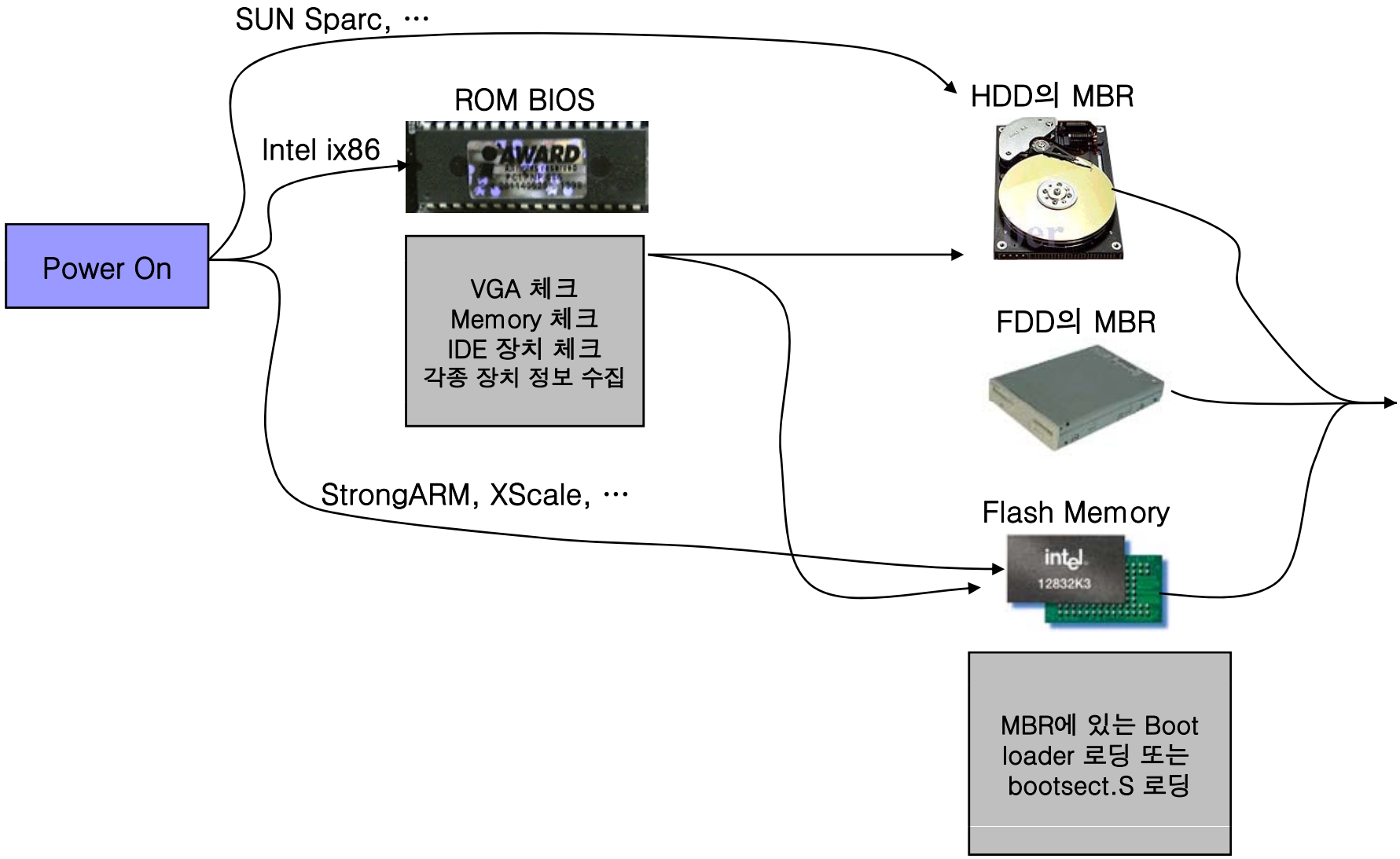
- Linux의 부팅 과정 이해
- 실행 파일과 링커 스크립트 관계 이해
- 부트로더 구조 및 원리 파악
- Blob을 통한 부트로더 소스 분석

Booting의 의미

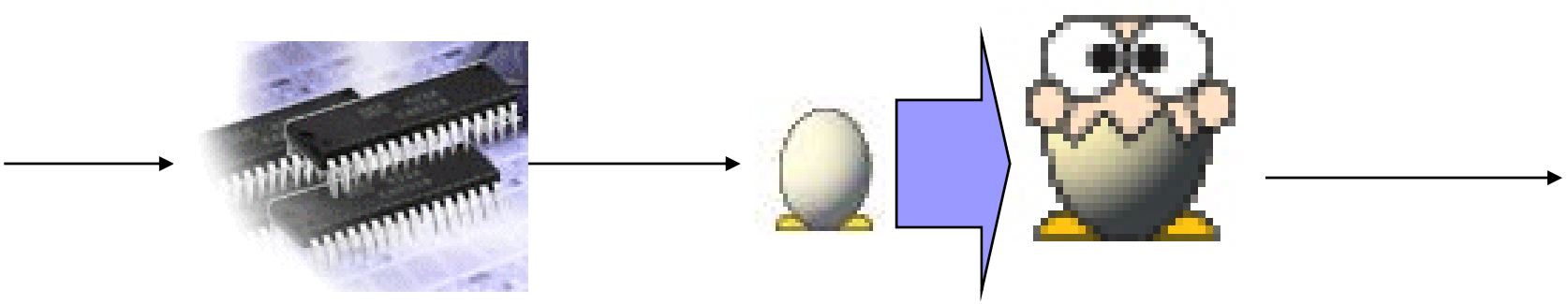
- Booting의 정의
  - ✓ Kernel이 메모리에 올려지고 하드웨어가 초기화 되어 바로 사용 가능한 상태로 만드는 과정을 부팅이라고 한다.

- Booting의 목적
  - ✓ processor 초기화
  - ✓ memory 점검 및 초기화
  - ✓ 각종 하드웨어 점검 및 초기화
  - ✓ kernel loading
  - ✓ kernel 자료구조 등록 및 초기화
  - ✓ 사용환경 조성

부팅 과정 도식도(1/5)


SUN Sparc, …
ROM BIOS
HDD의 MBR
Intel ix86
Power On
VGA 체크
Memory 체크
IDE 장치 체크
각종 장치 정보 수집
FDD의 MBR
StrongARM, XScale, …
Flash Memory
MBR에 있는 Boot loader 로딩 또는 bootsect.S 로딩

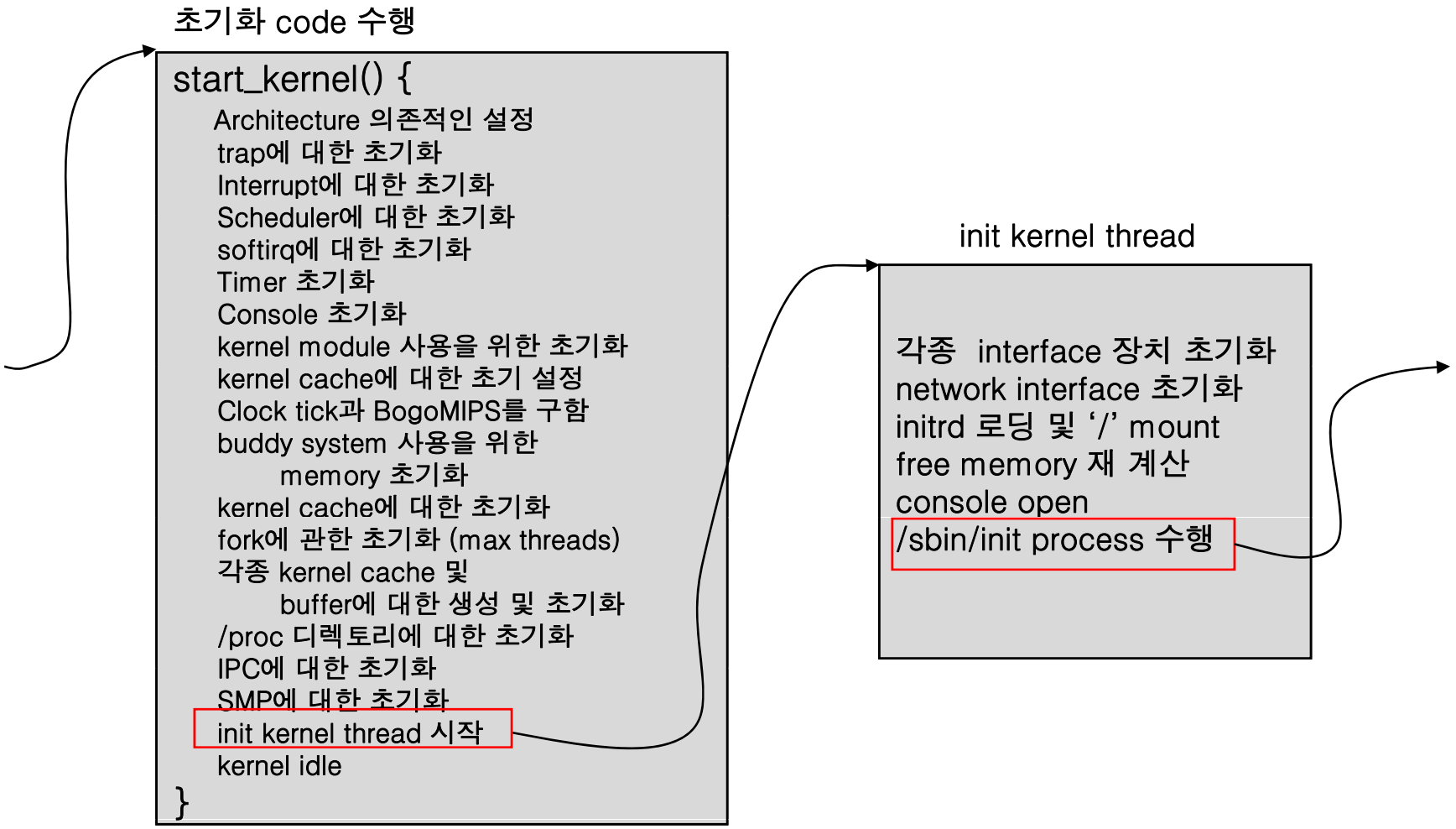
부팅 과정 도식도(2/5)



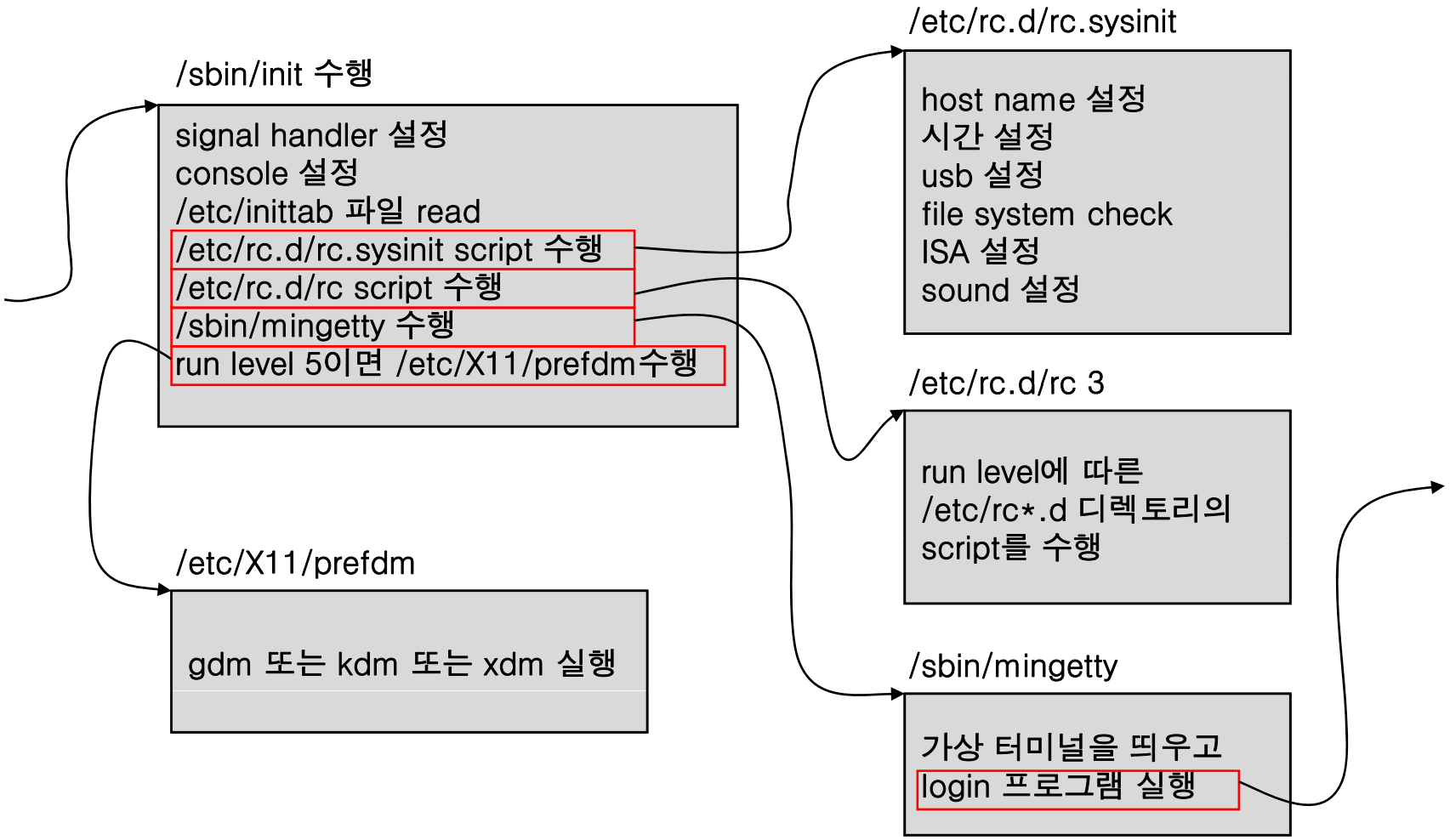
kernel을 main memory로 로딩

kernel 압축 해제

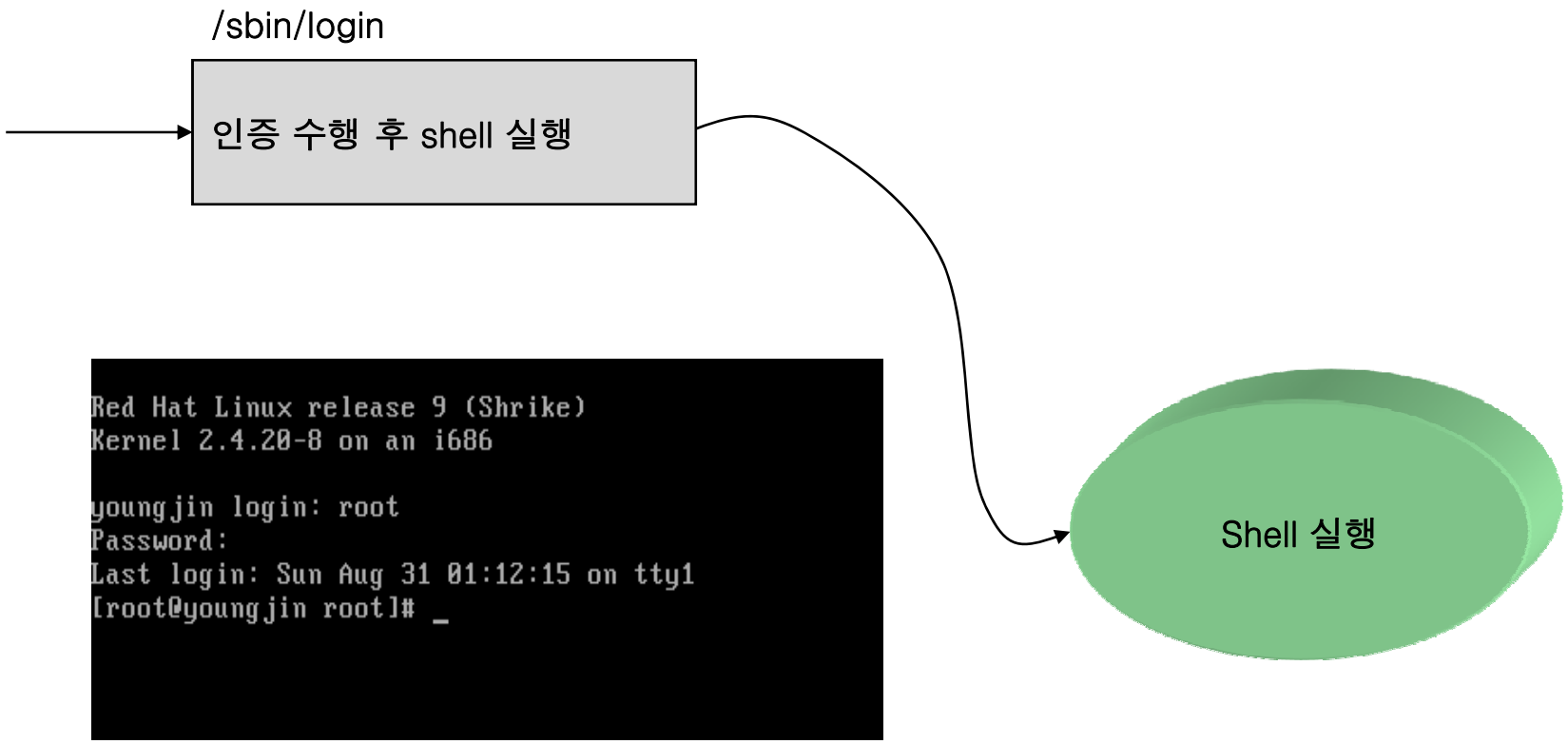
부팅 과정 도식도(3/5)


초기화 code 수행
start_kernel() {
Architecture 의존적인 설정
trap에 대한 초기화
Interrupt에 대한 초기화
Scheduler에 대한 초기화
softirq에 대한 초기화
Timer 초기화
Console 초기화
kernel module 사용을 위한 초기화
kernel cache에 대한 초기 설정
Clock tick과 BogoMIPS를 구함
buddy system 사용을 위한
memory 초기화
kernel cache에 대한 초기화
fork에 관한 초기화 (max threads)
각종 kernel cache 및
buffer에 대한 생성 및 초기화
/proc 디렉토리에 대한 초기화
IPC에 대한 초기화
SMP에 대한 초기화
init kernel thread 시작
kernel idle
}
init kernel thread
각종 interface 장치 초기화
network interface 초기화
initrd 로딩 및 '/' mount
free memory 재 계산
console open
/sbin/init process 수행

부팅 과정 도식도(4/5)

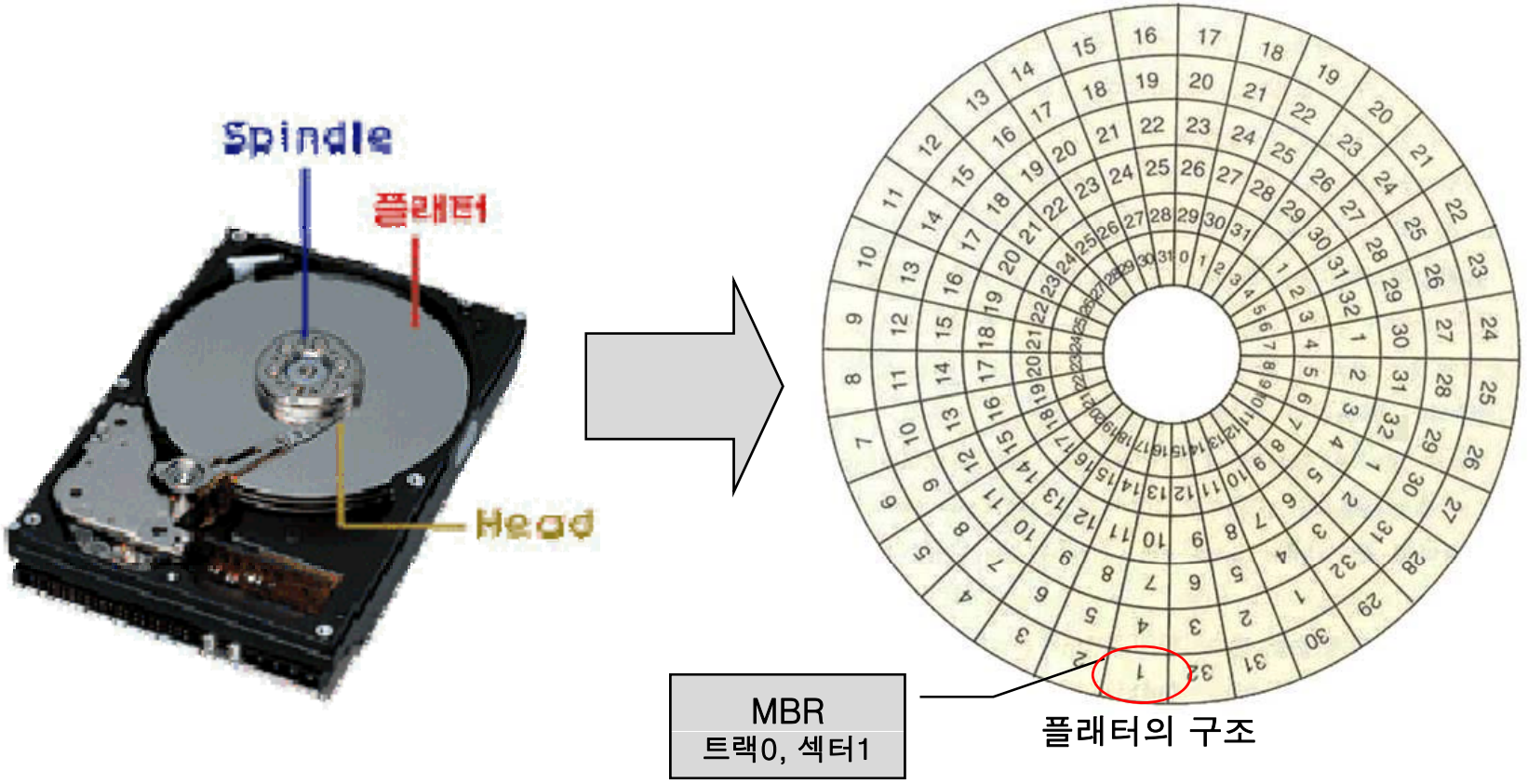

/sbin/init 수행
signal handler 설정
console 설정
/etc/inittab 파일 read
/etc/rc.d/rc.sysinit script 수행
/etc/rc.d/rc script 수행
/sbin/mingetty 수행
run level 5이면 /etc/X11/prefdm수행
/etc/rc.d/rc.sysinit
host name 설정
시간 설정
usb 설정
file system check
ISA 설정
sound 설정
/etc/rc.d/rc 3
run level에 따른
/etc/rc*.d 디렉토리의
script를 수행
/sbin/mingetty
가상 터미널을 띄우고
login 프로그램 실행
/etc/X11/prefdm
gdm 또는 kdm 또는 xdm 실행

부팅 과정 도식도(5/5)


/sbin/login
인증 수행 후 shell 실행
Red Hat Linux release 9 (Shrike)
Kernel 2.4.20-8 on an i686
youngjin login: root
Password:
Last login: Sun Aug 31 01:12:15 on tty1
[root@youngjin root]# _
Shell 실행

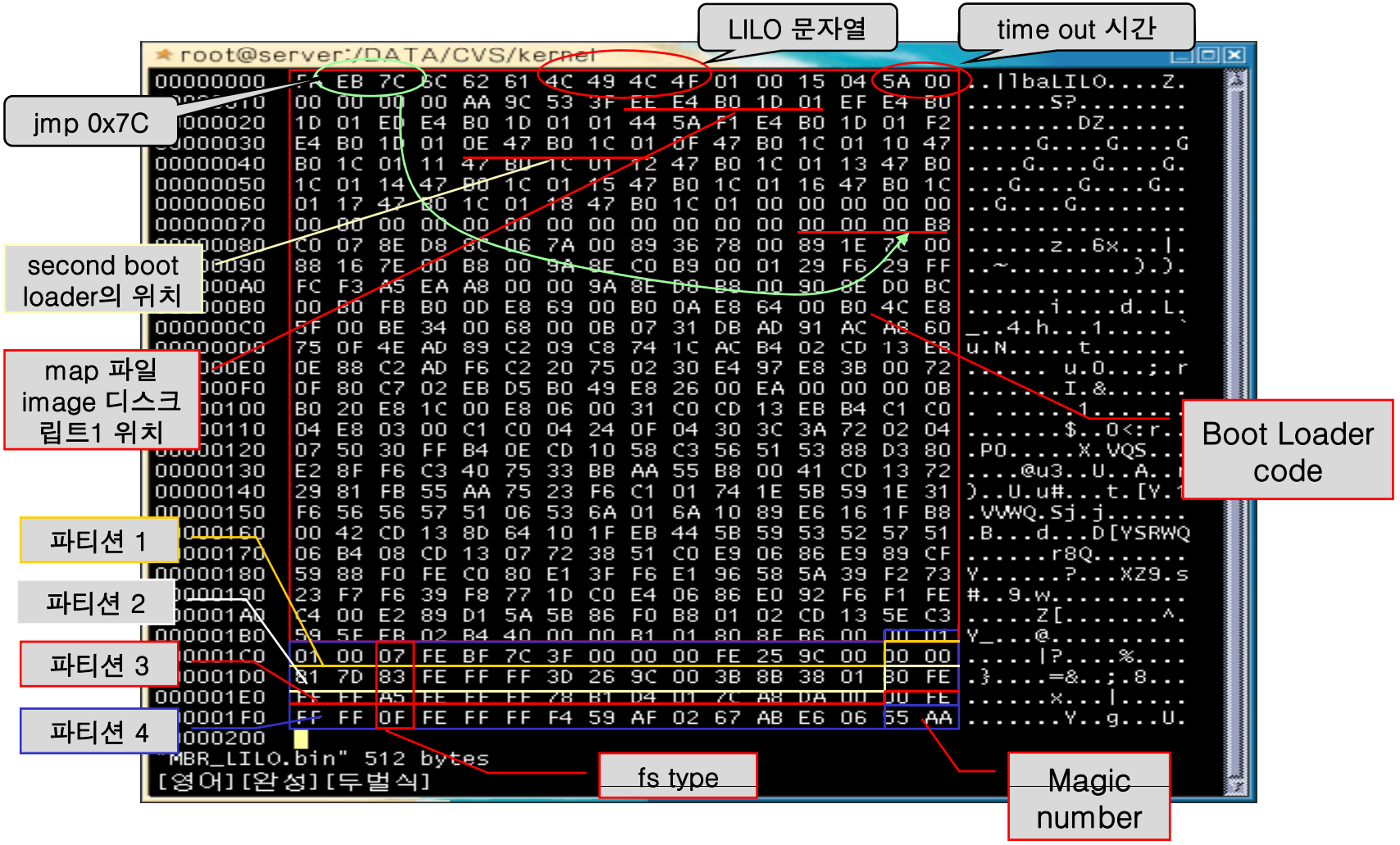
MBR의 이해(1/2)

- MBR의 이해
  - ✓ MBR이란 하드디스크로 부팅하기 위한 boot loader와 파티션 분할 정보, 부팅에 사용되는 실제 파티션 (ACTIVE PARTITION)에 대한 정보가 저장된 곳으로 하드디스크의 제일 바깥쪽에 위치한 공간으로(절대섹터0(Cylinder 0, Head 0, Sector 1), 크기:1sector(512byte)) 하드 디스크로 들어오는 관문이 되는 곳이다. MBR은 boot sector에 포함되나 모든 boot sector가 MBR은 아니다. boot sector는 각 파티션의 첫 번째 sector를 의미한다.



MBR의 이해(2/2)

MBR 이미지 (dd if=/dev/had of=MBR.img bs=1c count=512)



Bootloader의 역할과 종류

- Boot Loader의 역할
  - ✓ Kernel을 memory에 적재하고 제어를 kernel로 옮김
  - ✓ OS의 선택적 부팅
  - ✓ Kernel 다운로드를 제공하기도 한다
  - ✓ Embedded system을 위한 boot loader는 BIOS에서 해주는 하드웨어 초기화 작업 담당

- Boot Loader의 종류
  - ✓ LILO
    - 전통적인 linux boot loader이다. 일반적인 boot loader가 그렇듯 assembly로 짜여져 있고 크게 MBR에 들어가는 first.S와 /boot/boot.b로 만들어지는 second.S 두 부분으로 이루어져 있다
  - ✓ GRUB
    - 최근에 주목 받고 있는 boot loader로서 기능과 유연성 면에서 LILO보다 앞선다. GNU에서 만들었으며 뛰어난 shell interface를 제공한다
  - ✓ bootsector.S
    - Kernel에서 제공되는 boot loader로서 압축된 kernel의 제일 앞 512byte 공간을 차지하고 있으며 floppy 등으로 부팅할 때 사용되어지고 다른 boot loader로 부팅할 때는 건너 띄는 부분이다
  - ✓ Blob
    - ARM SA-11x0 architecture에서 사용하는 대표적인 boot loader로서 GNU GPL이여서 사용에 제한이 없고 serial을 통한 다운로드를 지원한다

Blob

Entry point 확인

- ~/src/start-ld-script

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
        . = 0x00000000;

        . = ALIGN(4);
        .text : { *(.text) }

        . = ALIGN(4);
        .rodata : { *(.rodata) }

        . = ALIGN(4);
        .data : { *(.data) }

        . = ALIGN(4);
        .got : { *(.got) }

        . = ALIGN(4);
        .bss : { *(.bss) }
}
```

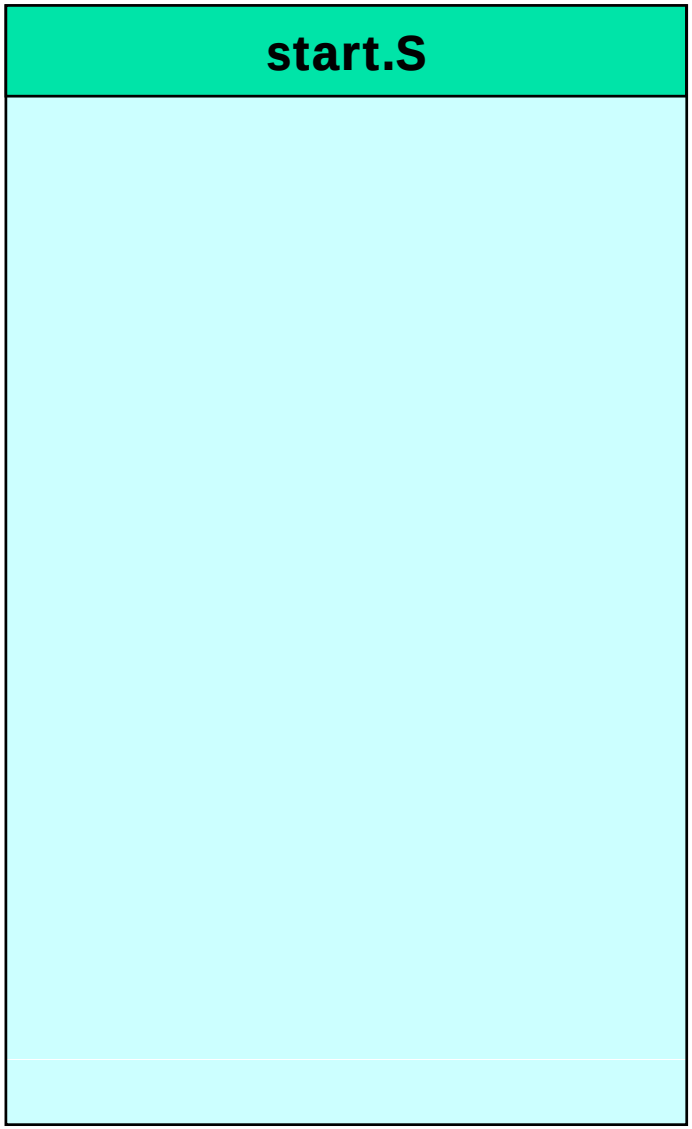
32bit ELF 포맷을 가지는 **little endian** 형태의 코드를 생성

이 파일이 컴파일된 뒤 **ARM**이라는 **Architecture**에서 실행됨을 알려줌

실행파일의 **entry point**가 **_start** 라는 레이블임을 알려줌

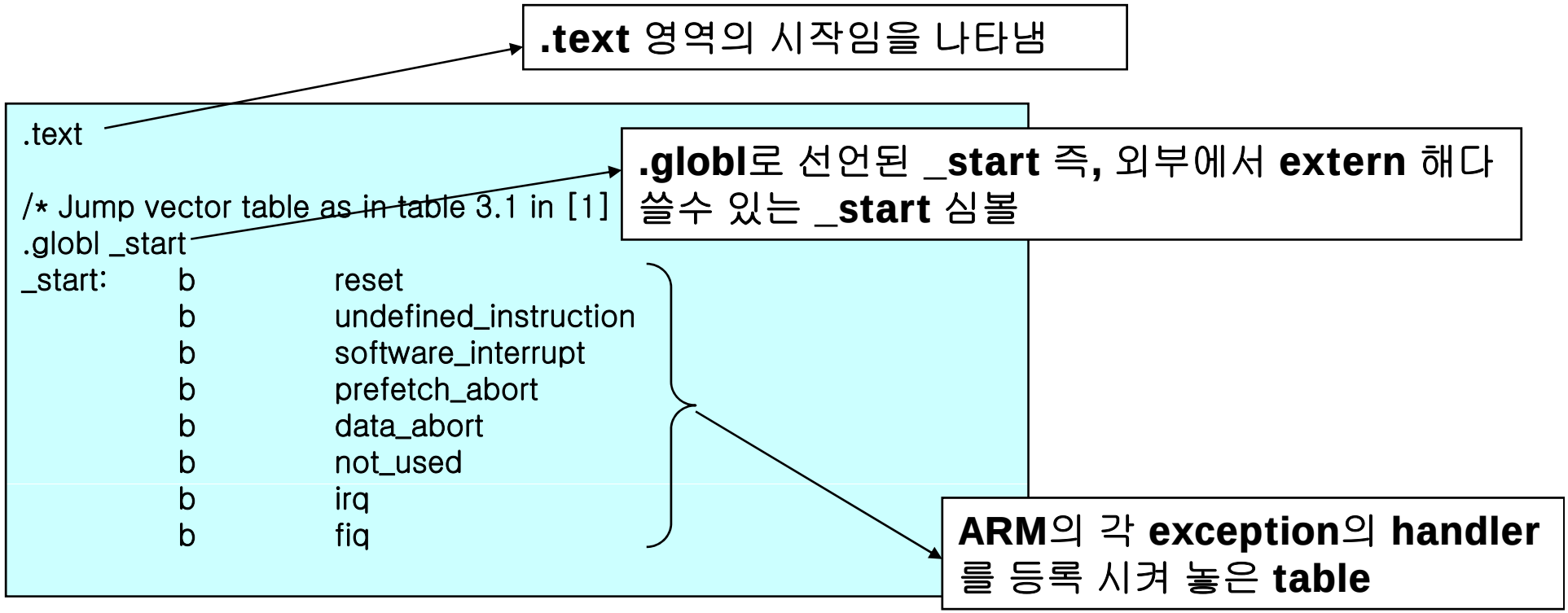
물리주소 **0x00000000** 번지에서 수행이 시작되며, 실행파일은 코드가 들어가 있는 **text**와 **rodata, data, got, bss**의 순서대로 **4byte** 단위로 구성됨을 의미

Memory 구성도



_start 레이블

- ~/src/start.S



reset 레이블

- ~/src/start.S

```
IC_BASE:        .word   0x90050000
#define ICMR    0x04

reset:
                /* First, mask **ALL** interrupts */
                ldr             r0, IC_BASE
                mov             r1, #0x00
                str             r1, [r0, #ICMR]
```

모든 인터럽트를 불허함

Interrupt Controller Registers		
0h 9005 0000	ICIP	Interrupt controller irq pending register.
0h 9005 0004	ICMR	Interrupt controller mask register.
0h 9005 0008	ICLR	Interrupt controller FIQ level register.
0h 9005 000C	ICCR	Interrupt controller control register.
0h 9005 0010	ICFP	Interrupt controller FIQ pending register.
0h 9005 0020	ICPR	Interrupt controller pending register.

reset 레이블

- ~/src/start.S

```
PWR_BASE:       .word   0x90020000
#define PSPR    0x08
#define PPCR    0x14

        /* switch CPU to correct speed */
        ldr     r0, PWR_BASE
        LDR     r1, cpuspeed
        str     r1, [r0, #PPCR]
        /* setup memory */
        bl      memsetup
        /* init LED */
        bl      ledinit
```

→ **CPU clock speed** 설정

Power Manager Registers		
0h 9002 0000	PMCR	Power manager control register.
0h 9002 0004	PSSR	Power manager sleep status register.
0h 9002 0008	PSPR	Power manager scratchpad register.
0h 9002 000C	PWER	Power manager wakeup enable register.
0h 9002 0010	PCFR	Power manager configuration register.
0h 9002 0014	PPCR	Power manager PLL configuration register.
0h 9002 0018	PGSR	Power manager GPIO sleep state register.
0h 9002 001C	POSR	Power manager oscillator status register.

reset 레이블

- ~/src/start.S

```
/* The initial CPU speed. Note that the SA11x0 CPUs can be safely overclocked:
 * 190 MHz CPUs are able to run at 221 MHz, 133 MHz CPUs can do 206 Mhz.
 */
#if (defined ASSABET) || (defined CLART) || (defined LART) ₩
            || (defined NESA) || (defined NESA)
cpuspeed: .long       0x0b        /* 221 MHz */
#elif defined SHANNON
cpuspeed: .long       0x09        /* 191.7 MHz */
#else
#warning "FIXME: Include code to use the correct clock spe
cpuspeed: .long       0x05        /* safe 133 MHz speed */
#endif
```

CCF 4:0	Core Clock Fre
	3.6864-MHz Crystal Oscillator
00000	59.0
00001	73.7
00010	88.5
00011	103.2
00100	118.0
00101	132.7
00110	147.5
00111	162.2
01000	176.9
01001	191.7
01010	206.4
01011	221.2
01100– 11111	Not supported.

memsetup 레이블

- ~/src/memsetup.S

```
.text

MEM_BASE:           .long       0xa0000000
MEM_START:          .long       0xc0000000
#define     MDCNFG  0x0
#define MDCAS0      0x04
#define MDCAS1      0x08
#define MDCAS2      0x0c
#define MCS0        0x10
#if (defined BRUTUS)
mdcas0:             .long       0xc71c703f
mdcas1:             .long       0xffc71c71
mdcas2:             .long       0xffffffff
mdcnfg:             .long       0x0334b22f
mcs0:               .long       0xfff8fff8
#endif

.globl memsetup
memsetup:
```

memsetup 레이블

- ~/src/memsetup.S

```
#elif defined USE_SA1110
        /* This part is actually for the Assabet only. If your board
         * uses other settings, you'll have to ifdef
         */
        /* Set up the SDRAM */
        mov     r1, #0xA0000000         /* MDCNFG base address */

        ldr     r2, =0xAAAAAA7F
        str     r2, [r1, #0x04]         /* MDCAS00 */
        str     r2, [r1, #0x20]         /* MDCAS20 */

        ldr     r2, =0xAAAAAAAA
        str     r2, [r1, #0x08]         /* MDCAS01 */
        str     r2, [r1, #0x24]         /* MDCAS21 */

        ldr     r2, =0xAAAAAAAA
        str     r2, [r1, #0x0C]         /* MDCAS02 */
        str     r2, [r1, #0x28]         /* MDCAS22 */

        ldr     r2, =0x4dbc0327         /* MDREFR */
        str     r2, [r1, #0x1C]

        ldr     r2, =0x72547254         /* MDCNFG */
        str     r2, [r1, #0x00]
```

MDCAS00, 20, 01, 21, 02, 22, MDREFR, MDCNFG register설정

memsetup 레이블

- ~/src/memsetup.S

```
        /* Issue read requests to disabled bank to start refresh */

        ldr     r1, =0xC0000000

.rept   8
        ldr     r0, [r1]
.endr

        mov     r1, #0xA0000000     /* MDCNFG base address */

        ldr     r2, =0x72547255     /* Enable the banks */
        str     r2, [r1, #0x00]     /* MDCNFG */
```

disable된 뱅크에 **refresh**가 일어나는 것을 막기위해 **8**번 반복하여 읽기 연산 수행

그런뒤 **MDCNFG**값을 **0x72547255**로 설정하여 **0/1 bank pair** 활성화

memsetup 레이블

- ~/src/memsetup.S

```
/* Static memory chip selects on Assabet: */

                ldr             r2, =0x4b90             /* MCS0 */
                orr             r2,r2,r2,lsl #16
                str             r2, [r1, #0x10]

                ldr             r2, =0x22212419         /* MCS1 */
                str             r2, [r1, #0x14]

                ldr             r2, =0x42196669         /* MCS2 */
                str             r2, [r1, #0x2C]

                ldr             r2, =0xafccafcc         /* SMCNFG */
                str             r2, [r1, #0x30]
/* Set up PCMCIA space */
                ldr             r2, =0x994a994a
                str             r2, [r1, #0x18]
/* All SDRAM memory settings should be ready to go... */
/* For best performance, should fill out remaining memory config regs: */

                /* Testing ,Chester */
                mov r3,#0x12000000
                mov r2,#0x5000              /* D9_LED on and D8_LED off */
                str r2,[r3]
                mov r4, #0x20000
gogogo2:
                subs r4, r4, #1
                bne gogogo2

                mov             pc, lr
```

Board specific한 **PCMCIA/memory** 설정을 수행

BL로 점프 해 왔으므로 **lr**값을 **PC**에 복원하여 원래 위치로 복귀

ledinit 레이블

- ~/src/ledasm.S

```
.text

LED:                    .long LED_GPIO
GPIO_BASE:              .long 0x90040000
#define     GPDR        0x00000004
#define GPSR            0x00000008
#define GPCR            0x0000000c




.globl ledinit
            /* initialise LED GPIO and turn LED on.
             * clobbers r0 and r1
             */
ledinit:
            ldr         r0, GPIO_BASE
            ldr         r1, LED
            str         r1, [r0, #GPDR]     /* LED GPIO is output */
            str         r1, [r0, #GPSR]     /* turn LED on */
            mov         pc, lr
```

LED 점등후 원래 루틴으로 복귀

reset 레이블

- ~/src/start.S

```
RST_BASE:           .word       0x90030000
#define RCSR        0x04

        /* check if this is a wake-up from sleep */
        ldr         r0, RST_BASE
        ldr         r1, [r0, #RCSR]
        and         r1, r1, #0x0f
        teq         r1, #0x08
        bne         normal_boot             /* no, continue booting */
```

SA11X0엔 **H/W reset, S/W reset, Watchdog reset, Sleep reset** 네 가지의 **reset**이 존재한다.
RCSR register는 어떤 이유로 **reset** 인터럽트가 발생했는가를 나타내며,
RSSR register는 **S/W reset**을 발생시키는 **register**이다.
따라서 **Blob**에선 **RCSR register**의 값을 읽어서 하위 네 비트 값을 통해 어떤 이유로 **reset**되었는지를 살펴 본뒤,

Sleep reset인 경우엔 대응하는 비트를 클리어 한뒤,
PSPR register값을 읽어 **r1**에 넣어주고,
이 값으로 **jump**함으로써 이전 상태로 복원하게 되며,
정상적인 **H/W reset**인 경우라면 **normal_boot** 레이블로
이동하게 된다.

0h 9003 0000 RSRR Write-Only

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																															SWR
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bits	Name	Description
0	SWR	Software reset. 0 – Do not invoke a software reset of the chip. 1 – Invoke a software reset of the chip. **Note**: This bit is self-resetting, and is automatically cleared several system clock cycles after it has been set.
31..1	—	Reserved

0h 9003 0004 RCSR Read/Write

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved																												SMR	WDR	SWR	HWR
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Bits	Name	Description
0	HWR	Hardware reset. 0 – Hardware reset has not occurred since the last time the CPU cleared this bit. 1 – Hardware reset has occurred since the last time the CPU cleared this bit.
1	SWR	Software reset. 0 – Software reset has not occurred since the last time the CPU cleared this bit. 1 – Software reset has occurred since the last time the CPU cleared this bit.
2	WDR	Watchdog reset. 0 – Watchdog reset has not occurred since the last time the CPU cleared this bit. 1 – Watchdog reset has occurred since the last time the CPU cleared this bit.
3	SMR	Sleep mode reset. 0 – Sleep mode reset has not occurred since the last time the CPU cleared this bit. 1 – Sleep mode reset has occurred since the last time the CPU cleared this bit.
31..4	—	Reserved

normal_boot 레이블

- ~/src/start.S

```
normal_boot:
        /* enable I-cache */
        mrc     p15, 0, r1, c1, c0, 0   @ read control reg
        orr     r1, r1, #0x1000         @ set Icache
        mcr     p15, 0, r1, c1, c0, 0   @ write it back
```

I-cache enable 작업 수행
control register값을 읽은 뒤 ➔ **Icache**를 **set**하고 ➔ 다시 덮어씀

normal_boot 레이블

- ~/src/start.S

```
normal_boot:
        /* enable I-cache */
        mrc     p15, 0, r1, c1, c0, 0   @ read control reg
        orr     r1, r1, #0x1000         @ set Icache
        mcr     p15, 0, r1, c1, c0, 0   @ write it back
```

I-cache enable 작업 수행
control register값을 읽은 뒤 ➔ **Icache**를 **set**하고 ➔ 다시 덮어씀

```
/* main memory starts at 0xc0000000 */
MEM_START:      .long   0xc0000000

        /* check the first 1MB  in increments of 4k */
        mov     r7, #0x1000
        mov     r6, r7, lsl #8          /* 4k << 2^8 = 1MB */
        ldr     r5, MEM_START
```

메모리 시작 번지로 부터 **1MB**를 테스트 한다.
r7에 **0x1000**을 넣고
좌로 **8**번 **shift**연산을 수행하여 **r6**에 **1M**를 넣은 후,
메모리의 시작 번지 주소인 **0xc000 0000** 을 **r5 register**에 넣는다.

normal_boot 레이블

- ~/src/start.S

```
mem_test_loop:
        mov     r0, r5
        bl      testram
        teq     r0, #1
        beq     badram

        add     r5, r5, r7
        subs    r6, r6, r7
        bne     mem_test_loop
```

그런 뒤 **(~/src/testmem.S)**파일 내에 정의 되어 있는 **testram**을 호출하여 메모리 테스트 수행

normal_boot 레이블

- ~/src/start.S

```
                /* the first megabyte is OK, so let's clear it */
                mov     r0, #((1024 * 1024) / (8 * 4))      /* 1MB in steps of 32 bytes */
                ldr     r1, MEM_START
                mov     r2, #0
                mov     r3, #0
                mov     r4, #0
                mov     r5, #0
                mov     r6, #0
                mov     r7, #0
                mov     r8, #0
                mov     r9, #0
clear_loop:
                stmia   r1!, {r2-r9}
                subs    r0, r0, #(8 * 4)
                bne     clear_loop
```

이상 없다면 루프를 돌면서 확인된 **1M**영역의 내용을 **0**으로 초기화 함.
r0를 **1MB/32** 값을 넣고, **r1**엔 메모리의 시작 주소를 넣는다.
stmia 명령어를 이용하여 **r1**이 가리키는 곳에 **r2-r9**의 값**(0)**을 넣어 주고 **r0**의 값을 **32** 만큼 감소 시킨다. 감소 시킨 값이 **0**이 아니면 계속 하여 루프를 돌면서 **1MB** 영역을 모두 **0**으로 만든다

normal_boot 레이블

- ~/src/start.S

```
/* get a clue where we are running, so we know what to copy */
and         r0, pc, #0xff000000    /* we don't care about the low bits */


/* relocate the second stage loader */
add         r2, r0, #(128 * 1024)          /* blob is 128kB */
add         r0, r0, #0x400                 /* skip first 1024 bytes */
ldr         r1, MEM_START
add         r1, r1, #0x400                 /* skip over here as well */
```

Blob 자신을 확인이 완료된 **RAM**으로 복사 하려 한다.
PC값을 이용해 어디서 부터 복사해야 하는지 결정한다.

Blob의 전체 **size**가 **128K**이므로 복사 작업을 끝마쳐야 하는 주소 즉, 현재 **PC**에 **128K**를 더한 값을 **r2**에 넣는다.

normal_boot 레이블

- ~/src/start.S

```
/* get a clue where we are running, so we know what to copy */
and         r0, pc, #0xff000000    /* we don't care about the low bits */


/* relocate the second stage loader */
add         r2, r0, #(128 * 1024)          /* blob is 128kB */
add         r0, r0, #0x400                 /* skip first 1024 bytes */
ldr         r1, MEM_START
add         r1, r1, #0x400                 /* skip over here as well */
```

r0에는 **PC + 0x400** 값을 넣는다. **r0**는 복사 작업에서 **source**의 위치를 나타내는데 이는 현재 수행되고 있는 **start.S** 외의 다른 파일의 컴파일에 관계가 있는 **rest-ld-script**에 **.text**의 시작이 **0xc0000400** 번지로 지정되어 있기때문이다.
다음으로 **r1**에는 복사 작업의 **destination** 값을 넣어 준다.
따라서 복사작업에서 **target** 주소를 나타내는 **r1**도 **0xc0000400**을 가리키도록 하여 **rest-ld-script**로 링킹된 **binary**들이 지장없이 수행되도록 한다.

normal_boot 레이블

- ~/src/start.S

```
copy_loop:
        ldmia   r0!, {r3-r10}
        stmia   r1!, {r3-r10}
        cmp     r0, r2
        ble     copy_loop
```

r0가 가리키고 있는 곳의 내용을 **r3~r10**에 담고
이 내용을 **r1**이 가리키고 있는 곳으로 저장한다.
이 작업은 **source**의 끝에 해당되는 **r2**와
ldmia 명령어를 통해 자동으로 증가 되고 있는 **r0**값이 같아 질때까지 수행된다.

```
        /* turn off the LED. if it stays off it is an indication that
         * we didn't make it into the C code
         */
        bl led_off
```

디버그 정보 출력을 위해 **led**를 끈다

normal_boot 레이블

- ~/src/start.S

```
/* set up the stack pointer */
ldr         r0, MEM_START
add         r1, r0, #(1024 * 1024)
sub         sp, r1, #0x04


/* blob is copied to ram, so jump to it */
add         r0, r0, #0x400
mov         pc, r0
```

Stack pointer를 **0xc0000000 + 1M – 0x04** 위치로 설정 한 후
PC를 방금 복사 해 놓은 **RAM**상의 **Blob**를 가리키게 한다.
따라서 **0xc0000400** 위치에 있는 실행 파일로 제어가 넘어간다.

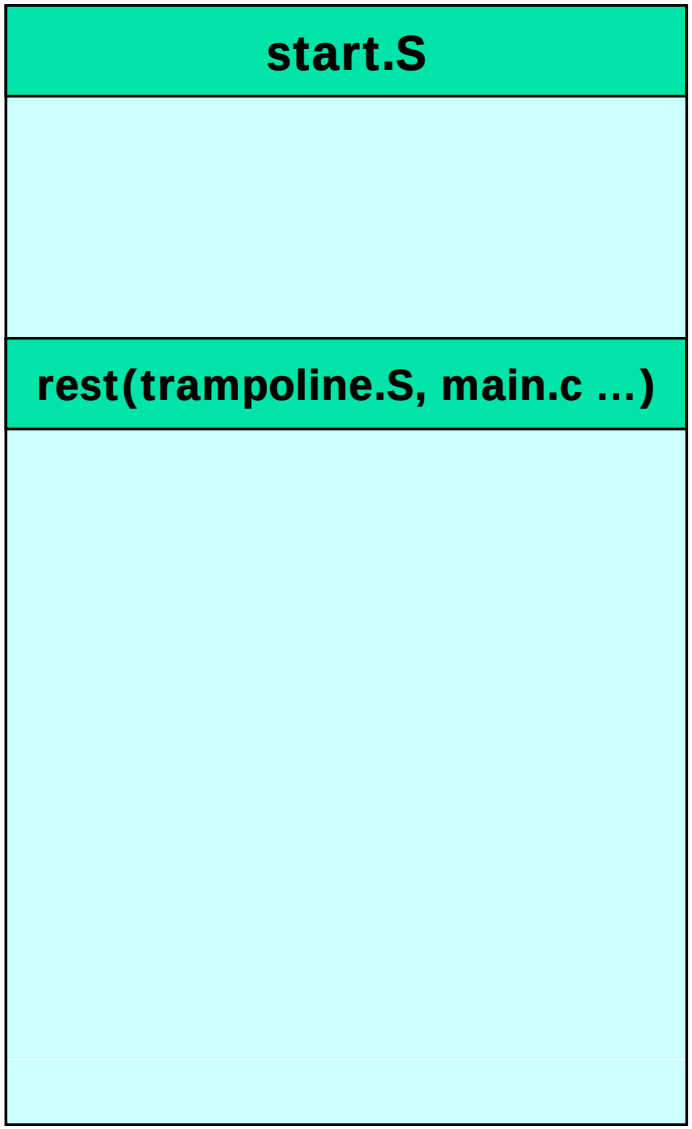
rest-ld-script

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_trampoline)
SECTIONS
{
        . = 0xc0000400;

        . = ALIGN(4);
        .text : { *(.text) }

        . = ALIGN(4);
        .rodata : { *(.rodata) }

        . = ALIGN(4);
        .data : { *(.data) }

        . = ALIGN(4);
        .got : { *(.got) }

        . = ALIGN(4);
        .bss : { *(.bss) }
}
```

실행파일의 **entry point**가 **_trampoline** 이라는 레이블임을 알려줌

물리주소 **0xc0000400** 번지에서 수행이 시작되며, 실행파일은 코드가 들어가 있는 **text**와 **rodata, data, got, bss**의 순서대로 **4byte** 단위로 구성됨을 의미

Memory 구성도



0x 0000 0000

0x c000 0400

_trampoline 레이블

- ~/src/trampoline.S

```
.text

.globl _trampoline
_trampoline:
        bl      main
        /* if main ever returns we just call it again */
        b       _trampoline
```

main으로 분기하는 역할 만을 담당함

만약 **main**에서 리턴한다면 다시 분기

main 함수

- ~/src/main.c

```
int main(void)
{
        u32 blockSize = 0x00800000;
        int numRead = 0;
        char commandline[128];
        int i;
        int retval = 0;

        /* Turn the LED on again, so we can see that we safely made it
         * into C code.
         */
        led_on();

        /* We really want to be able to communicate, so initialise the
         * serial port at 9k6 (which works good for terminals)
         */
        SerialInit(baud9k6);
        TimerInit();

        /* Print the required GPL string */
        SerialOutputString("₩nConsider yourself LARTed!₩n₩n");
        SerialOutputString(PACKAGE " version " VERSION  "₩n"
                           "Copyright (C) 1999 2000 2001 " "Jan-Derk Bakker and Erik Mouw₩n"
                           "Copyright (C) 2000 "  "Johan Pouwelse₩n");
        SerialOutputString(PACKAGE " comes with ABSOLUTELY NO WARRANTY; "
                           "read the GNU GPL for details.₩n");
        SerialOutputString("This is free software, and you are welcome " "to redistribute it₩n");
        SerialOutputString("under certain conditions; " "read the GNU GPL for details.₩n");
```

LED를 켠후, 시리얼 포트를 초기화한 뒤

타이머를 초기화 하고, **msg**출력

SerialInit() 함수

- ~/src/serial.c

```
void SerialInit(eBauds baudrate)
{
        /* Theory of operations:
         * - Flush the output buffer
         * - switch receiver and transmitter off
         * - clear all sticky bits in control register 3
         * - set the port to sensible defaults (no break, no interrupts,
         *   no parity, 8 databits, 1 stopbit, transmitter and receiver
         *   enabled
         * - set the baudrate to the requested value
         * - turn the receiver and transmitter back on
         */

#if defined USE_SERIAL1
        while(Ser1UTSR1 & UTSR1_TBY) {
        }

        Ser1UTCR3 = 0x00;
        Ser1UTSR0 = 0xff;
        Ser1UTCR0 = ( UTCR0_1StpBit | UTCR0_8BitData );
        Ser1UTCR1 = 0;
        Ser1UTCR2 = (u32)baudrate;
        Ser1UTCR3 = ( UTCR3_RXE | UTCR3_TXE );
#elif defined USE_SERIAL3
        while(Ser3UTSR1 & UTSR1_TBY) {
        }

        Ser3UTCR3 = 0x00;
        Ser3UTSR0 = 0xff;
        Ser3UTCR0 = ( UTCR0_1StpBit | UTCR0_8BitData );
        Ser3UTCR1 = 0;
        Ser3UTCR2 = (u32)baudrate;
        Ser3UTCR3 = ( UTCR3_RXE | UTCR3_TXE );
#else
#error "Configuration error: No serial port used at all!"
#endif
}
```

TimerInit() 함수

- ~/src/time.c

```
void TimerInit(void)
{
        /* clear counter */
        OSCR = 0;

        /* we don't want to be interrupted */
        OIER = 0;

        /* wait until OSCR > 0 */
        while(OSCR == 0)
                ;

        /* clear match register 0 */
        OSMR0 = 0;

        /* clear match bit for OSMR0 */
        OSSR = OSSR_M0;

        numOverflows = 0;
}
```

1개의 **OSCR(Operating System Counter Register)(up-counter register)**과 **4**개의 **OSMR(Operating System Match Register)** 존재

OSCR이 **OSMR**과 일치되면 **OSSR(Operating System Status Register)**의 해당 비트가 **set**되고**, INT**가 **enable**된 경우라면 **INT**발생됨

main 함수

- ~/src/main.c

```
/* get the amount of memory */
get_memory_map();

/* initialise status */
blob_status.kernelSize = 0;
blob_status.kernelType = fromFlash;
blob_status.ramdiskSize = 0;
blob_status.ramdiskType = fromFlash;
blob_status.blockSize = blockSize;
blob_status.downloadSpeed = baud115k2;
```

get_memory_map()함수를 호출하여 시스템의 메모리 양을 결정하고

blob_status 구조체에 변수를 초기화 함

get_memory_map() 함수

- ~/src/memory.c

```
#define NUM_MEM_AREAS   (32)
typedef struct {
        u32 start;
        u32 len;
        int used;
} memory_area_t;
extern memory_area_t memory_map[NUM_MEM_AREAS]
```

```
/* memory start and end */
#define MEMORY_START                    (0xc0000000)
#define MEMORY_END                      (0xe0000000)
#define TEST_BLOCK_SIZE                 (1024 * 1024)

void get_memory_map(void)
{
        u32 addr;
        int i;

        /* init */
        for(i = 0; i < NUM_MEM_AREAS; i++)
                memory_map[i].used = 0;

        /* first write a 0 to all memory locations */
        for(addr = MEMORY_START; addr < MEMORY_END; addr += TEST_BLOCK_SIZE)
                * (u32 *)addr = 0;
```

memory의 상태를 저장할 구조체의 사용중을 표시하는 **used**필드를 **clear** 하고

실제 물리 메모리에 **MegaB**단위로 건너뛰면서 **0**을 쓴다

get_memory_map() 함수

- ~/src/memory.c

```
        i = 0;
        for(addr = MEMORY_START; addr < MEMORY_END; addr += TEST_BLOCK_SIZE) {
                if(testram(addr) == 0) {
                        /* yes, memory */
                        if(* (u32 *)addr != 0) { /* alias? */
                                if(memory_map[i].used)
                                        i++;
                                continue;
                        }

                        /* not an alias, write
                        * (u32 *)addr = addr
                        /* does this start a n
                        if(memory_map[i].us
                                memory_map[i].start = addr;
                                memory_map[i].len = TEST_BLOCK_SIZE;
                                memory_map[i].used = 1;
                        } else {
                                memory_map[i].len += TEST_BLOCK_SIZE;
                        }
                } else {
                        /* no memory here */
                        if(memory_map[i].used == 1)
                                i++;
                }
        }
        SerialOutputString("Memory map
        for(i = 0; i < NUM_MEM_AREAS;
                if(memory_map[i].used) {
                        SerialOutputString("  0x");
                        SerialOutputHex(memory_map[i].len);
                        SerialOutputString(" @ 0x");
                        SerialOutputHex(memory_map[i].start);
                        SerialOutputString(" (");
                        SerialOutputDec(memory_map[i].len / (1024 * 1024));
                        SerialOutputString(" MB)₩n");
                }
        }
}
```

~/src/testmem.S내의 **testram**을 호출하여 메모리를 검사한다.

정상적으로 검사가 수행되면 **0**을 리턴한다.

새로운 **area**가 아니라면 이전 **area**에 **size**만 더해준다.

새로운 **area**라면 구조체를 적절히 셋팅한다

그런 뒤 확인된 메모리 양을 화면에 출력

blob_status 구조체 설명

```
typedef enum {
        fromFlash = 0,
        fromDownload = 1
} block_source_t;

typedef struct {
        int kernelSize;
        block_source_t kernelType;

        int ramdiskSize;
        block_source_t ramdiskType;

        int blobSize;
        block_source_t blobType;

        u32 blockSize;

        eBauds downloadSpeed;
} blob_status_t;

blob_status_t blob_status;
```

커널 이미지의 크기
커널을 어디서 가져오는 지를 정의

Ramdisk의 크기
Ramdisk를 어디서 가져오는 지를 정의

Blob 자체의 크기
Blob를 어디서 가져오는 지를 정의

하나의 **Block**의 크기를 정의

Downloading speed를 정의

main 함수

- ~/src/main.c

```
/* Load kernel and ramdisk from flash to RAM */
        Reload("blob");
        Reload("kernel");
        Reload("ramdisk");
```

Reload()함수를 이용하여 **blob, kernel, ramdisk**를 메인 메모리로 끌고 올라옴

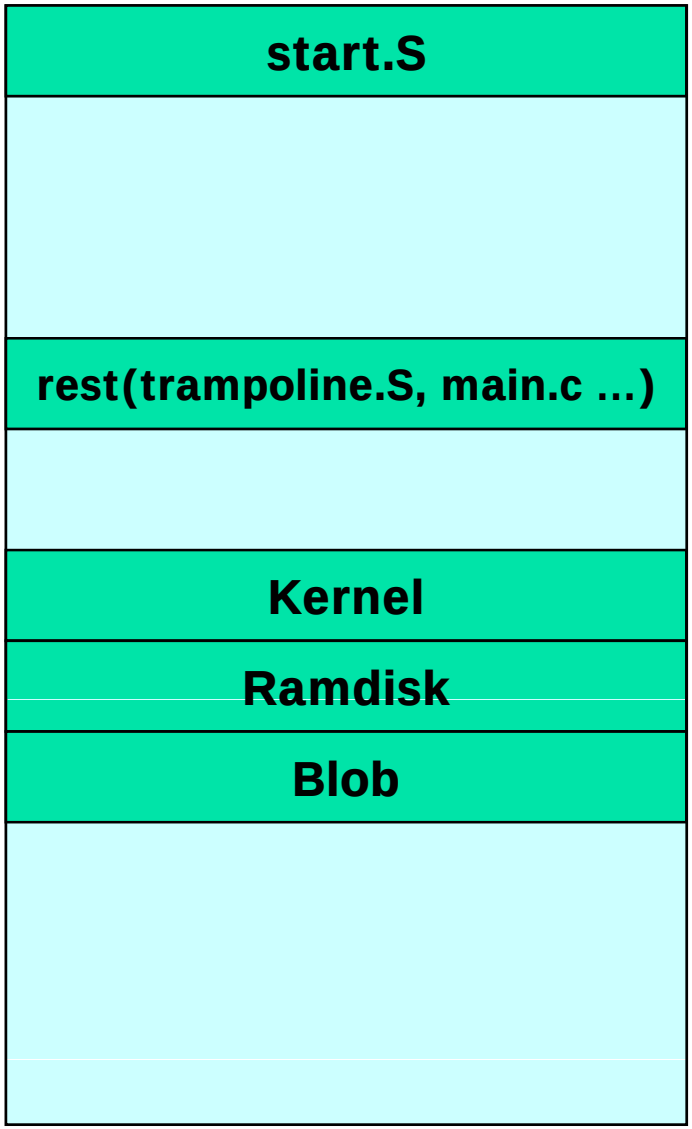
Reload 함수

- ~/src/main.c

```
void Reload(char *commandline)
{
	u32 *src = 0;
	u32 *dst = 0;
	int numWords;

	if(MyStrNCmp(commandline, "blob", 4) == 0) {
		src = (u32 *)BLOB_RAM_BASE;
		dst = (u32 *)BLOB_START;
		numWords = BLOB_LEN / 4;
		blob_status.blobSize = 0;
		blob_status.blobType = fromFlash;
		SerialOutputString("Loading blob from flash ");
	} else if(MyStrNCmp(commandline, "kernel", 6) == 0) {
		src = (u32 *)KERNEL_RAM_BASE;
		dst = (u32 *)KERNEL_START;
		numWords = KERNEL_LEN / 4;
		blob_status.kernelSize = 0;
		blob_status.kernelType = fromFlash;
		SerialOutputString("Loading kernel from flash ");
	} else if(MyStrNCmp(commandline, "ramdisk", 7) == 0) {
		src = (u32 *)RAMDISK_RAM_BASE;
		dst = (u32 *)INITRD_START;
		numWords = INITRD_LEN / 4;
		blob_status.ramdiskSize = 0;
		blob_status.ramdiskType = fromFlash;
		SerialOutputString("Loading ramdisk from flash ");
	} else {
		SerialOutputString("*** Don't know how to reload ₩"");
		SerialOutputString(commandline);
		SerialOutputString("₩"₩n");
		return;
	}

	MyMemCpy(src, dst, numWords);
	SerialOutputString(" done₩n");
}
```

MyStrNCmp()함수를 이용하여
인자로 넘어온 이미지를
Blob인 경우 **0xc100 0000** 번지에
Kernel인 경우 **0xc000 8000** 번지에
Ramdisk인 경우 **0xc080 0000** 번지에
올린다.

Memory 구성도


start.S
rest(trampoline.S, main.c …)
Kernel
Ramdisk
Blob
0x 0000 0000
0x c000 0400
0x c000 8000
0x c080 0000
0x c100 0000

main 함수

- ~/src/main.c

```
/* wait 10 seconds before starting autoboot */
        SerialOutputString("Autoboot in progress, press any key to stop ");
        for(i = 0; i < 10; i++) {
                SerialOutputByte('.');

                retval = SerialInputBlock(commandline, 1, 1);

                if(retval > 0)
                        break;
        }

        /* no key was pressed, so proceed booting the kernel */
        if(retval == 0) {
                commandline[0] = '₩0';
                boot_linux(commandline);
        }
```

10초간 기다린 뒤 별도의 입력이 없었다면 리눅스를 부팅 시킨다

main 함수

- ~/src/main.c

```
void boot_linux(char *commandline)
{
        register u32 i;
        void (*theKernel)(int zero, int arch) = (void (*)(int, int))KERNEL_RAM_BASE

        setup_start_tag();
        setup_memory_tags();
        setup_commandline_tag(commandline);
        setup_initrd_tag();
        setup_ramdisk_tag();
        setup_end_tag();

        /* we assume that the kernel is in place */
        SerialOutputString("₩nStarting kernel ...₩n₩n");

        /* turn off I-cache */
        asm ("mrc p15, 0, %0, c1, c0, 0": "=r" (i));
        i &= ~0x1000;
        asm ("mcr p15, 0, %0, c1, c0, 0": : "r" (i));

        /* flush I-cache */
        asm ("mcr p15, 0, %0, c7, c5, 0": : "r" (i));

        theKernel(0, ARCH_NUMBER);

        SerialOutputString("Hey, the kernel returned! This should not happen.₩n");
}
```

커널이 사용할 **BOOT_PARAMS**를 채워서 메모리 **0xc000 0100**에 올려놓고

I-cache를 끄고, **flush**시킨뒤

첫번째 **arg**에는 **0**을
두번째 **arg**에는 **ARCH_NUMBER**를 넣고
제어를 넘긴다.

tag구조체

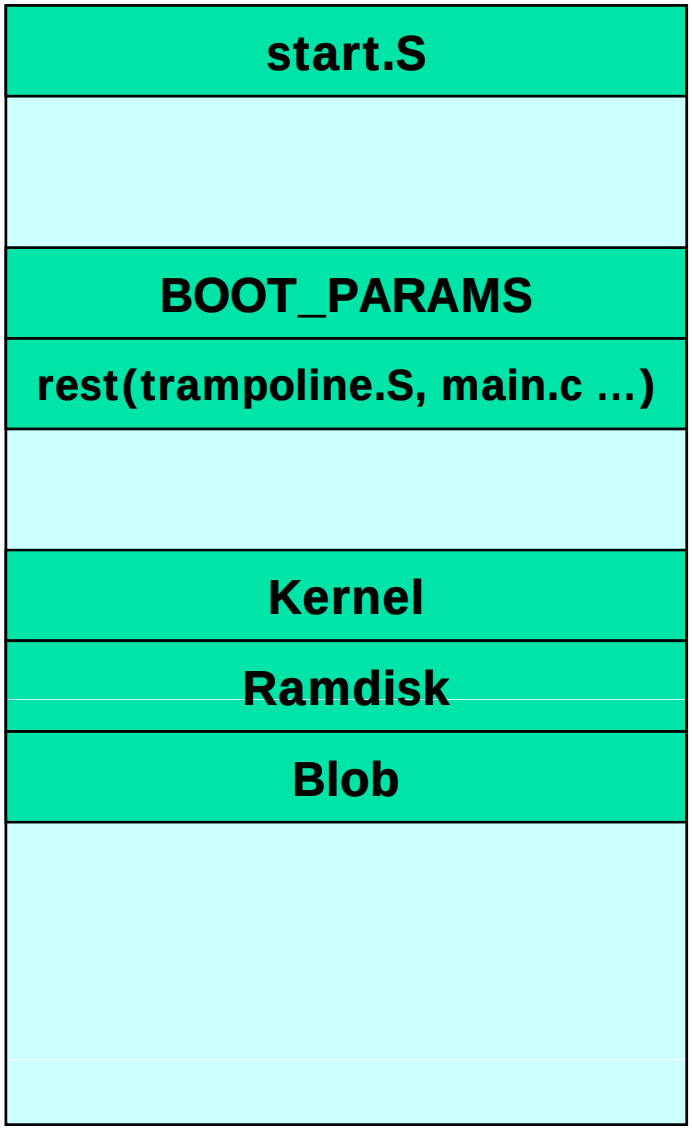
- linux/include/asm-arm/setup.h

```
struct tag_header {
        u32 size;
        u32 tag;
};

struct tag {
        struct tag_header hdr;
        union {
                struct tag_core         core;
                struct tag_mem32        mem;
                struct tag_videotext    videotext;
                struct tag_ramdisk      ramdisk;
                struct tag_initrd       initrd;
                struct tag_serialnr     serialnr;
                struct tag_revision     revision;
                struct tag_videolfb     videolfb;
                struct tag_cmdline      cmdline;

                /*
                 * Acorn specific
                 */
                struct tag_acorn        acorn;

                /*
                 * DC21285 specific
                 */
                struct tag_memclk       memclk;
        } u;
};
```

tag의 크기와 **magic number**

메모리의 크기와 시작 주소
비디오 램의 위치와 **resolution**
ramdisk의 옵션, 크기 및 시작 주소
Initial RAM disk의 크기와 시작 주소
Serial의 개수
Revision 번호
Video Frame Buffer의 설정 사항

Memory 구성도


start.S
BOOT_PARAMS
rest(trampoline.S, main.c …)
Kernel
Ramdisk
Blob
0x 0000 0000
0x c000 0100
0x c000 0400
0x c000 8000
0x c080 0000
0x c100 0000