# 부트로더 동작원리 분석

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

# Target Board가 생기면...

- Schematic
- Chip manual
- Bootloader
  - ✓ blob, u-boot, ...
- Kernel
- Rootfs
  - ✓ ramdisk, cramfs, jffs, ...

# 강의 목표

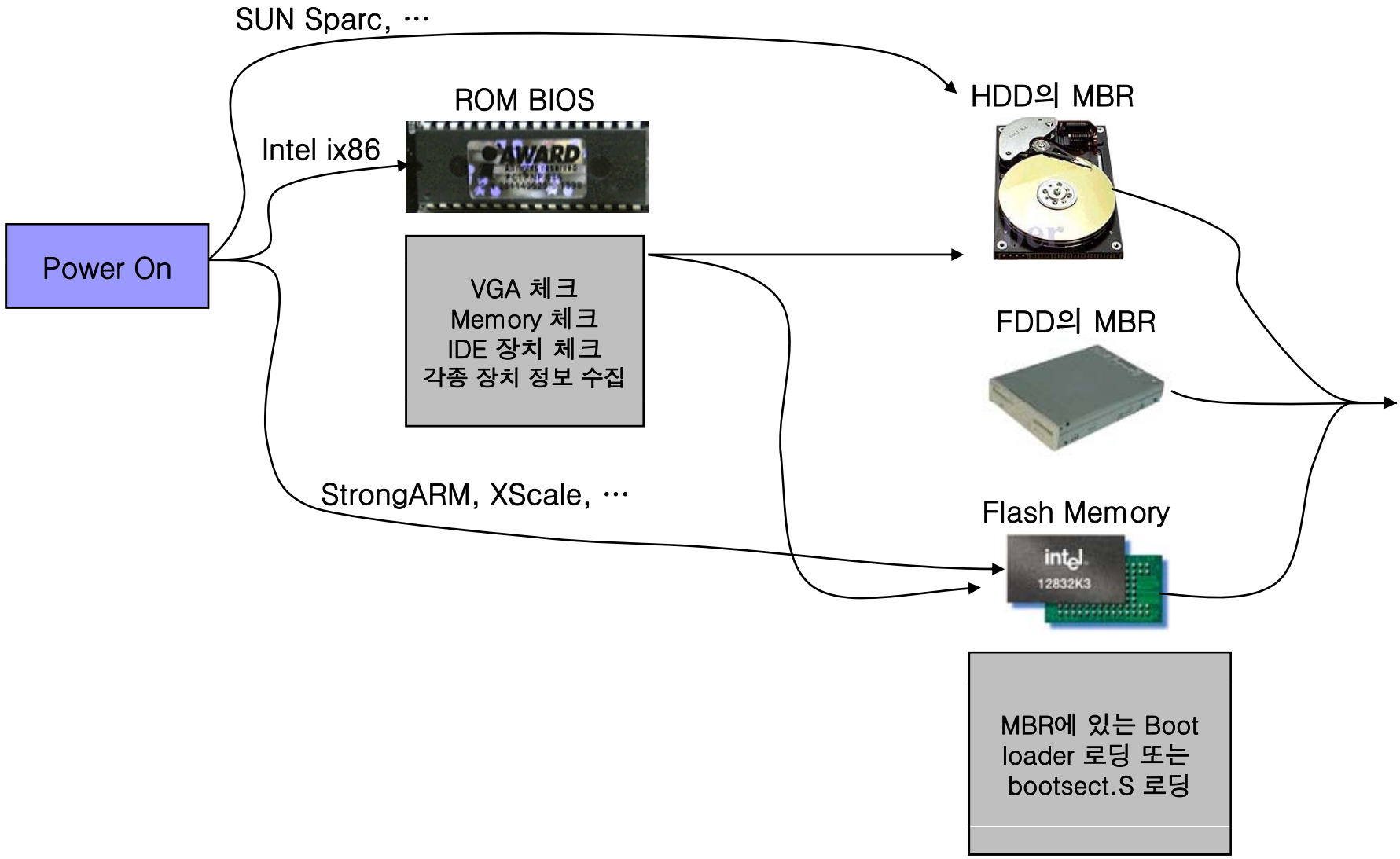
- Linux의 부팅 과정 이해
- 실행 파일과 링커 스크립트 관계 이해
- 부트로더 구조 및 원리 파악
- Blob을 통한 부트로더 소스 분석

# Booting의 의미

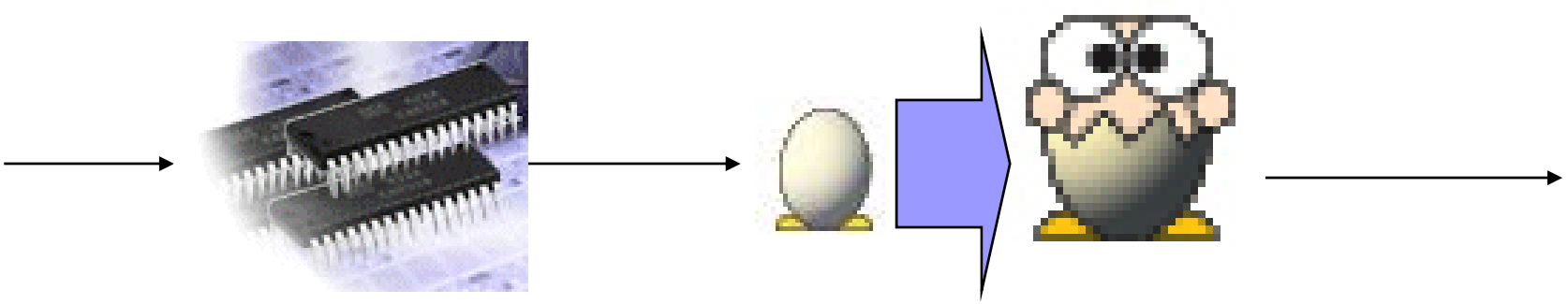
- Booting의 정의
  - ✓ Kernel이 메모리에 올려지고 하드웨어가 초기화 되어 바로 사용 가능한 상태로 만드는 과정을 부팅이라고 한다.

- 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

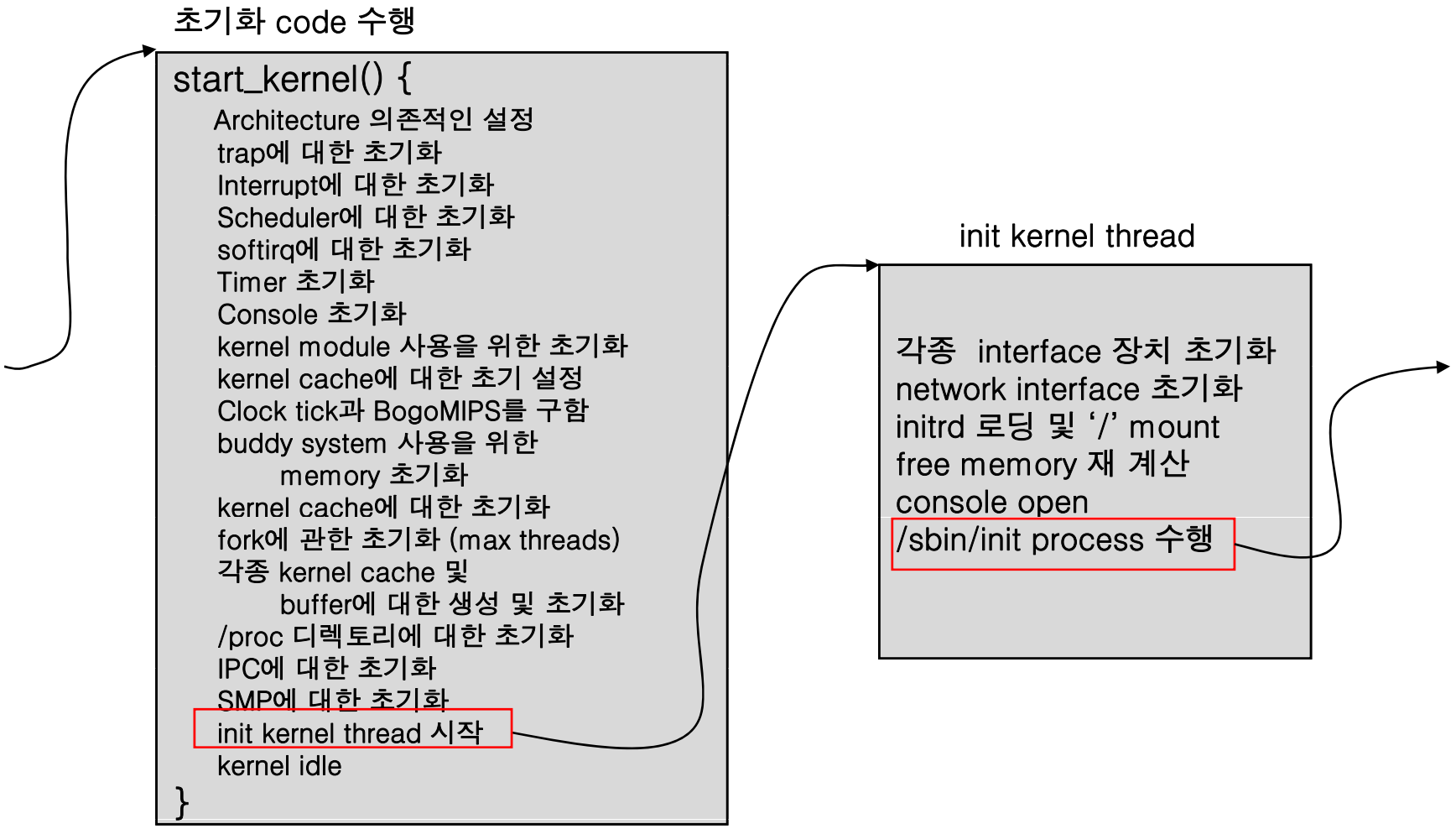
# 부팅 과정 도식도(2/5)



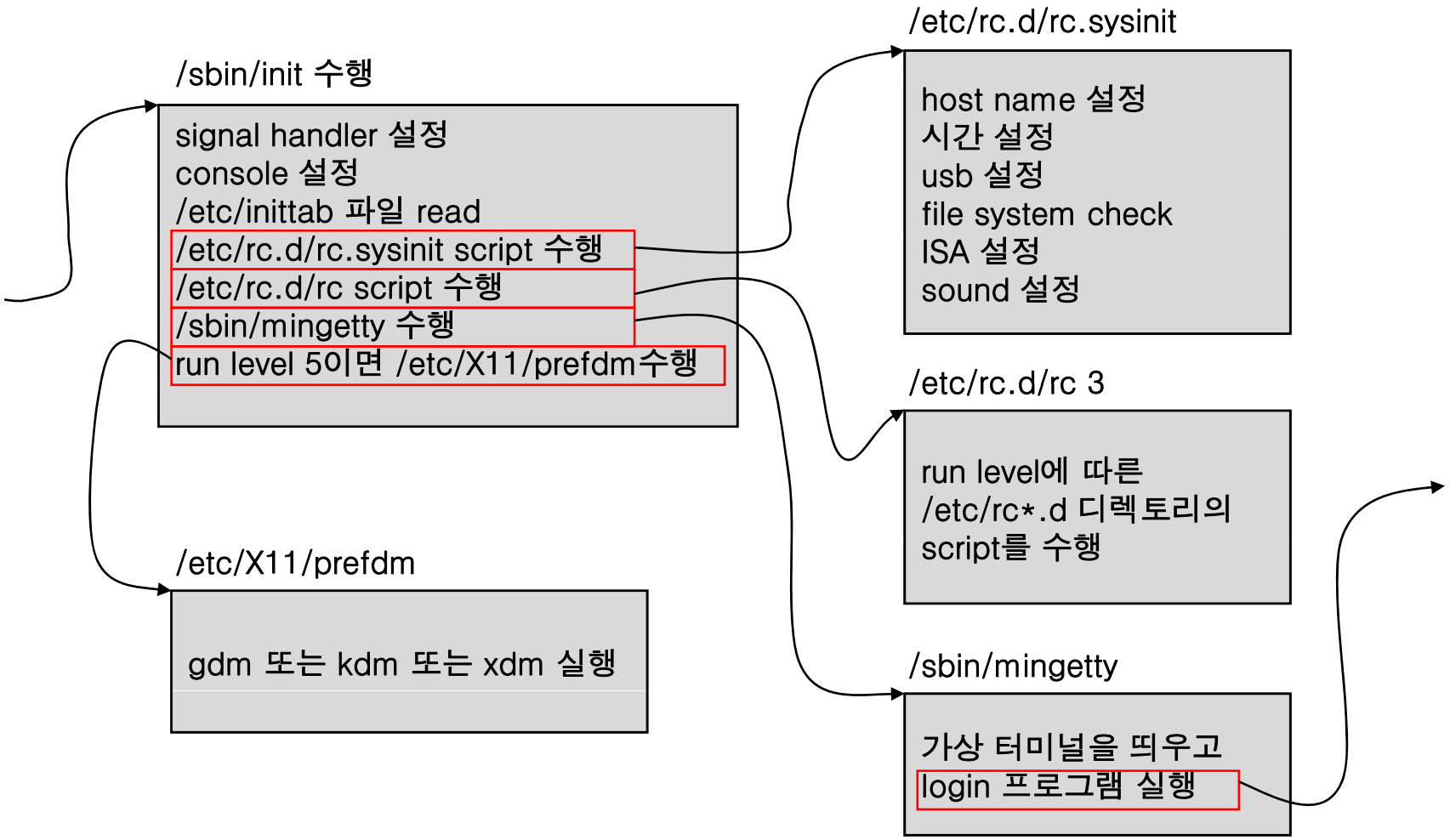
kernel을 main memory로 로딩

kernel 압축 해제

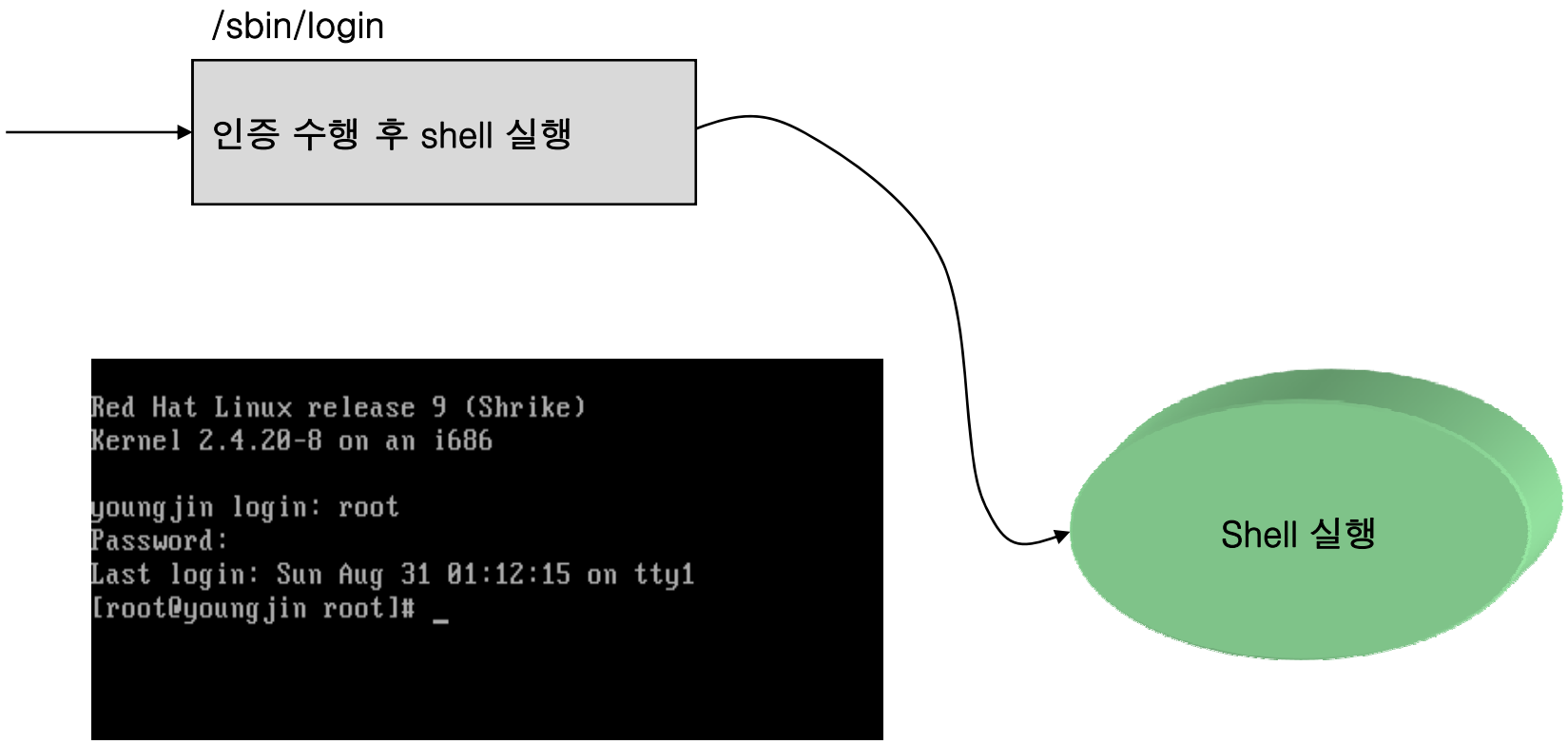
# 부팅 과정 도식도(3/5)


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

# 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
  - ✓ Blob
    - http://sourceforge.net/projects/blob
    - ARM SA-11x0 architecture에서 사용하는 대표적인 boot loader로서 GNU GPL이여서 사용에 제한이 없고 serial을 통한 다운로드를 지원한다
  - ✓ u-boot
    - http://www.denx.de/wiki/U-Boot

# Blob

# Entry point 확인

- ~/src/blob/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) }
}
```

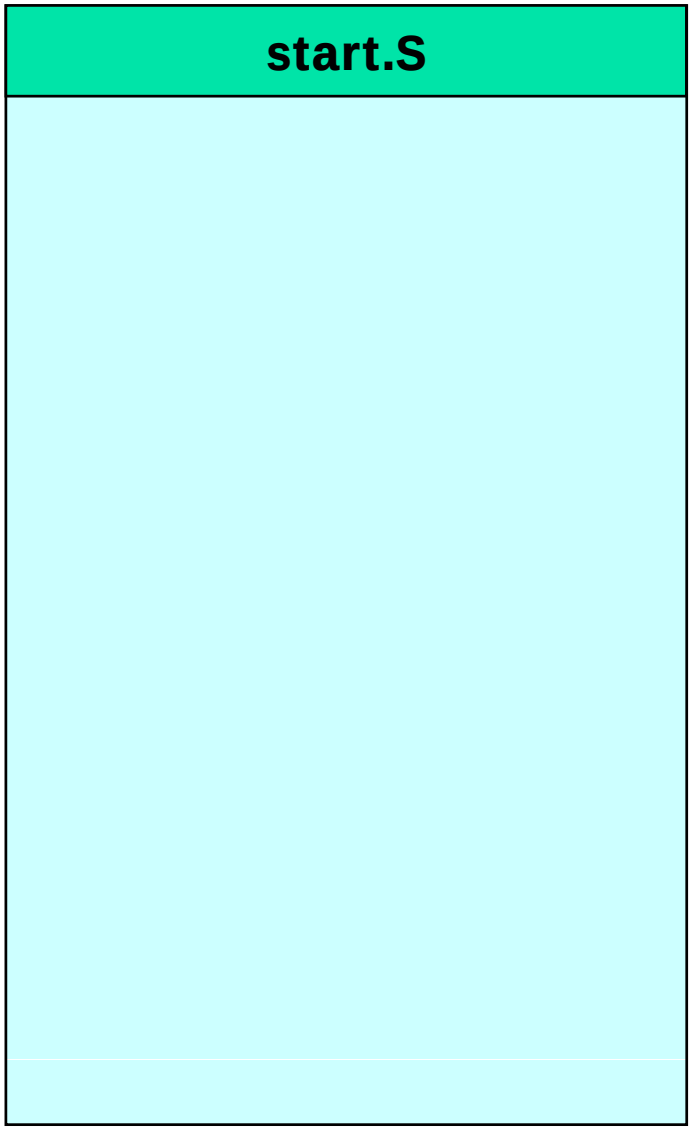
**32bit ELF** 포맷을 가지는 **little endian** 형태의 코드를 생성

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

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

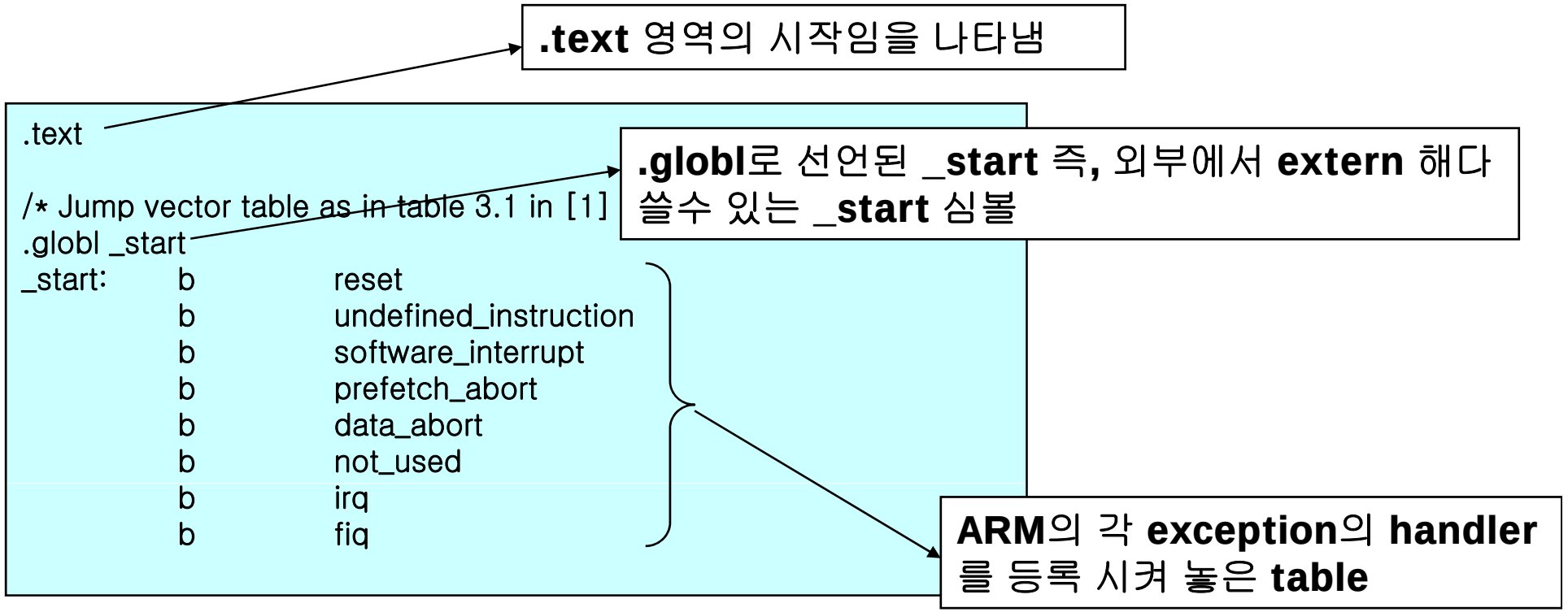
물리주소 **0x00000000** 번지에서 수행이 시작되며, 실행파일은 코드가 들어가 있는 **text**와 **rodata, data, got, bss**의 순서대로 **4byte** 단위로 구성됨을 의미

# Memory 구성도



# _start 레이블

- ~/src/blob/start.S



# reset 레이블

- ~/src/blob/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/blob/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/blob/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. |

# ledinit 레이블

- ~/src/blob/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/blob/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 */
```

**SA11X0**엔 **H/W reset, S/W reset, Watchdog reset, Sleep reset** 네 가지의 **reset**이 존재한다.
**RCSR register**는 어떤 이유로 **reset** 인터럽트가 발생했는가를 나타내며,
**RSSR register**는 **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 reset**인 경우엔 대응하는 비트를 클리어 한뒤,
**PSPR register**값을 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.<br>0 – Do not invoke a software reset of the chip.<br>1 – Invoke a software reset of the chip.<br>**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.<br>0 – Hardware reset has not occurred since the last time the CPU cleared this bit.<br>1 – Hardware reset has occurred since the last time the CPU cleared this bit. |
| 1 | SWR | Software reset.<br>0 – Software reset has not occurred since the last time the CPU cleared this bit.<br>1 – Software reset has occurred since the last time the CPU cleared this bit. |
| 2 | WDR | Watchdog reset.<br>0 – Watchdog reset has not occurred since the last time the CPU cleared this bit.<br>1 – Watchdog reset has occurred since the last time the CPU cleared this bit. |
| 3 | SMR | Sleep mode reset.<br>0 – Sleep mode reset has not occurred since the last time the CPU cleared this bit.<br>1 – Sleep mode reset has occurred since the last time the CPU cleared this bit. |
| 31..4 | — | Reserved |

# normal_boot 레이블

- ~/src/blob/start.S

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

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
```

메모리 시작 번지로 부터 **1MB**를 테스트 한다.
**r7**에 **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/blob/start.S

```
relocate:
        adr     r0, _start
        /* relocate the second stage loader */
        add     r2, r0, #(64 * 1024)    /* blob maximum size is 64kB */
        add     r0, r0, #0x400          /* skip first 1024 bytes */
        ldr     r1, BLOB_START

        /* r0 = source address
         * r1 = target address
         * r2 = source end address
         */
copy_loop:
        ldmia   r0!, {r3-r10}
        stmia   r1!, {r3-r10}
        cmp     r0, r2
        ble     copy_loop

        bl led_off
        /* blob is copied to ram, so jump to it */
        ldr     r0, BLOB_START
        mov     pc, r0
```

**Blob을 RAM상으로 복사한 뒤, 해당 주소로 점프**

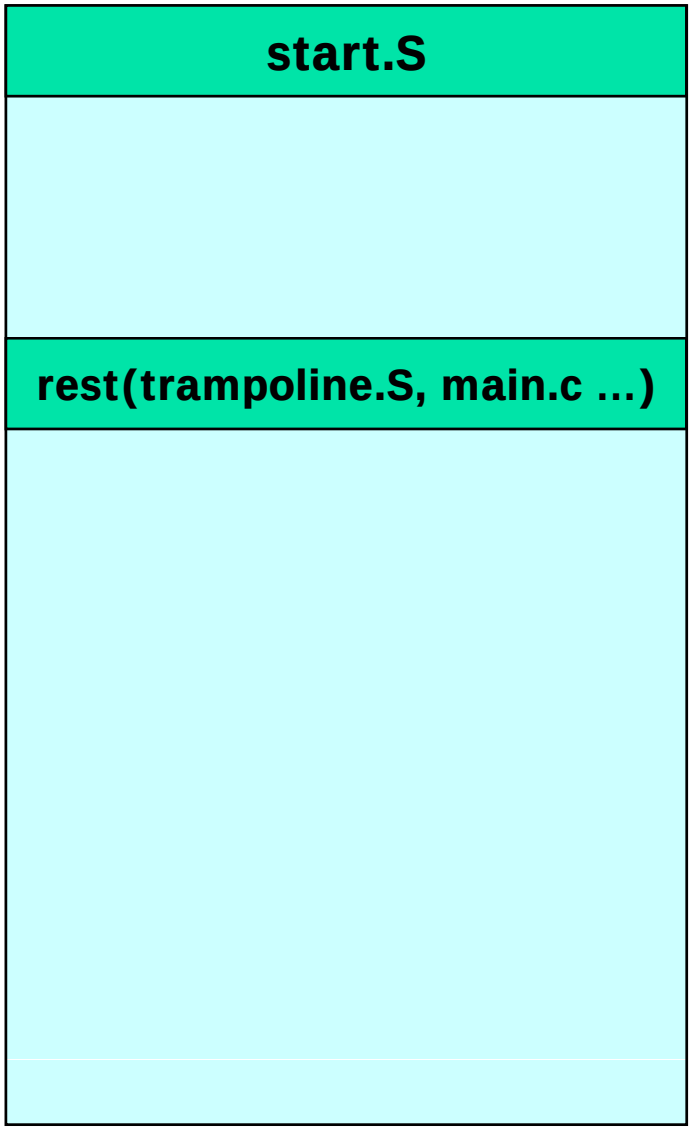
# rest-ld-script

```
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/blob/trampoline.S

```
.text
.globl _trampoline
_trampoline:
        /* clear the BSS section */
        ldr     r1, bss_start
        ldr     r0, bss_end
        sub     r0, r0, r1
        /* r1 = start address */
        /* r0 = #number of bytes */
        mov     r2, #0
clear_bss:
        stmia   r1!, {r2}
        subs    r0, r0, #4
        bne     clear_bss
        /* setup the stack pointer */
        ldr     r0, stack_end
        sub     sp, r0, #4
        /* jump to C code */
        bl      main
        /* if main ever returns we just call it again */
        b       _trampoline
```

**BSS** 영역 **clear** 후**, stack pointer** 설정하고**,**

**C**로 작성된 **main** 함수로 제어를 넘김

# main 함수

- ~/src/blob/main.c

```
int main(void)
{
        int numRead = 0;
        char commandline[MAX_COMMANDLINE_LENGTH];
        int i;
        int retval = 0;
#ifdef PARAM_START
        u32 conf;
#endif
        /* call subsystems */
        init_subsystems();
        /* initialise status */
        blob_status.paramType = fromFlash;
        blob_status.kernelType = fromFlash;
        blob_status.ramdiskType = fromFlash;
        blob_status.downloadSpeed = baud_115200;
        blob_status.terminalSpeed = baud_9600;
        blob_status.load_ramdisk = 1;
        blob_status.cmdline[0] = '₩0';
        blob_status.boot_delay = 10;
        /* call serial_init() because the default 9k6 speed might not
           be what the user requested */
#if defined(H3600) || defined(SHANNON) || defined(IDR) || defined(BADGE4) || defined(JORNADA720)
        blob_status.terminalSpeed = baud_115200; /* DEBUG */
#endif
#if defined(PT_SYSTEM3)
        blob_status.terminalSpeed = baud_38400;
#endif
        serial_init(blob_status.terminalSpeed);
```

시리얼 포트를 초기화한 뒤

**blob_status** 구조체에 변수를 초기화 함

# main 함수

- ~/src/blob/main.c

```
        /* Print the required GPL string */
        SerialOutputString("₩nConsider yourself LARTed!₩n₩n");
        SerialOutputString(version_str);
        SerialOutputString("Copyright (C) 1999 2000 2001 "
                           "Jan-Derk Bakker and Erik Mouw₩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");
        /* get the amount of memory */
        get_memory_map();
        print_elf_sections();
        /* Parse all the tags in the paramater block */
#ifdef PARAM_START
        parse_ptags((void *) PARAM_START, &conf);
#endif
```

**get_memory_map()**함수를 호출하여 시스템의 메모리 양을 결정하고

**blob_status** 구조체에 변수를 초기화 함

# blob_status 구조체 설명

```
typedef enum {
        fromFlash = 0,
        fromDownload = 1
} block_source_t;
typedef struct {
        int kernelSize;
        block_source_t kernelType;
        u32 kernel_md5_digest[4];
        int paramSize;
        block_source_t paramType;
        u32 param_md5_digest[4];
        int ramdiskSize;
        block_source_t ramdiskType;
        u32 ramdisk_md5_digest[4];
        int blobSize;
        block_source_t blobType;
        u32 blob_md5_digest[4];
        serial_baud_t downloadSpeed;
        serial_baud_t terminalSpeed;
        int load_ramdisk;
        int boot_delay;
        char cmdline[COMMAND_LINE_SIZE];
} blob_status_t;
blob_status_t blob_status;
```

커널 이미지의 크기
커널을 어디서 가져오는 지를 정의

**Ramdisk**의 크기
**Ramdisk**를 어디서 가져오는 지를 정의

**Blob** 자체의 크기
**Blob**를 어디서 가져오는 지를 정의

# main 함수

- ~/src/blob/main.c

```
do_reload("blob");
do_reload("kernel");
if(blob_status.load_ramdisk)
        do_reload("ramdisk");

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

        retval = SerialInputBlock(commandline, 1, 1);

        if(retval > 0)
                break;
}

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

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

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

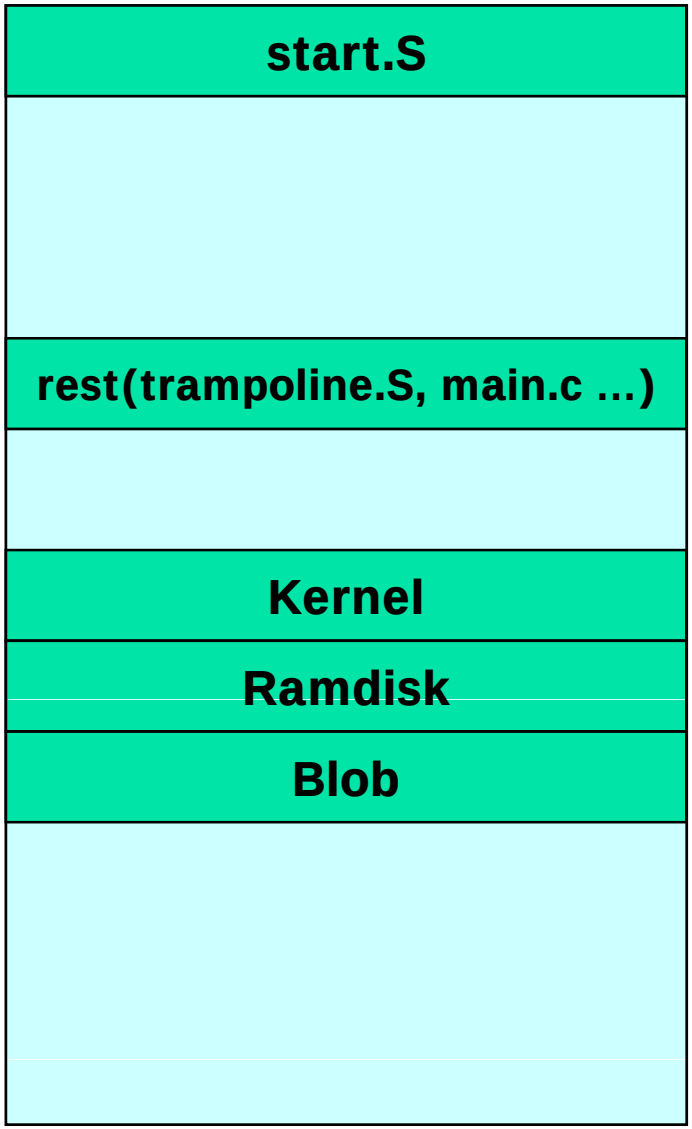
# Reload 함수

- ~/src/blob/main.c

```
void do_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
**Blob**인 경우 **0xc100 0000** 번지에
**Kernel**인 경우 **0xc000 8000** 번지에
**Ramdisk**인 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/blob/main.c

```
static int boot_linux(int argc, char *argv[])
{
        void (*theKernel)(int zero, int arch) = (void (*)(int, int))KERNEL_RAM_BASE

        setup_start_tag();
        setup_memory_tags();
        setup_commandline_tag(argc, argv);
        setup_initrd_tag();
        setup_ramdisk_tag();
        setup_end_tag();

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

        /* disable subsystems that want to be disabled before kernel boot */
        exit_subsystems();

        /* start kernel */
        theKernel(0, ARCH_NUMBER);

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

        return 0;
}
```

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

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

# tag구조체

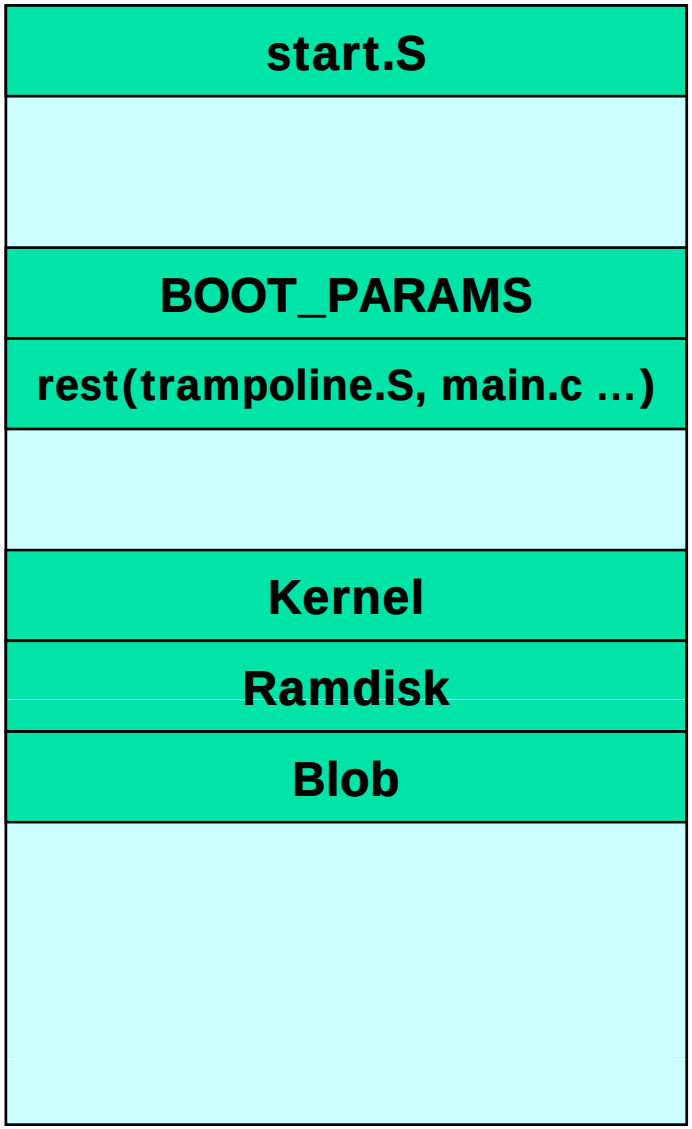
- linux/include/asm-arm/setup.h

```
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;
};
```

**tag**의 크기와 **magic number**

메모리의 크기와 시작 주소
비디오 램의 위치와 **resolution**
**ramdisk**의 옵션, 크기 및 시작 주소
**Initial RAM disk**의 크기와 시작 주소
**Serial**의 개수
**Revision** 번호
**Video Frame Buffer**의 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