由于没有CONFIG_NO_BOOTMEM=y，所以在memblock与buddy allocator之间没有boot memory manager的参与。

```
start_kernel()
     |
     |
    \|/
mm_init()
     |
     |
    \|/
mem_init()
```

在mem_init()中完成内存管理从memblock到buddy allocator的过渡。

在mem_init()以前，所有memory的状况都记录在struct memblock **memblock**这个global variable中

in mm/memblock.c

```
struct memblock memblock __initdata_memblock = {
	.memory.regions		= memblock_memory_init_regions,
	.memory.cnt		= 1,	/* empty dummy entry */
	.memory.max		= INIT_MEMBLOCK_REGIONS,

	.reserved.regions	= memblock_reserved_init_regions,
```

```
	.reserved.cnt		= 1,	/* empty dummy entry */
	.reserved.max		= INIT_MEMBLOCK_REGIONS,

#ifdef CONFIG_HAVE_MEMBLOCK_PHYS_MAP
	.physmem.regions	= memblock_physmem_init_regions,
	.physmem.cnt		= 1,	/* empty dummy entry */
	.physmem.max		= INIT_PHYSMEM_REGIONS,
#endif

	.bottom_up		= false,
	.current_limit		= MEMBLOCK_ALLOC_ANYWHERE,
};
```

1. 所有可利用的memory都在memblock.memory.regions[]这个数组中，memblock.memory.cnt是这个array中的有效项数。每一项都是一个memblock_region。
memblock.memory.regions[memblock.memory.cnt]

```
struct memblock_region {
	phys_addr_t base;
	phys_addr_t size;
	unsigned long flags;
#ifdef CONFIG_HAVE_MEMBLOCK_NODE_MAP
	int nid;
#endif
};
```

base and size指定了一块region(physical memory).

并且这些entry都是按memblock_region的base排序的（从小到大）。

2. 所有已经被reserved,即不能作为free memory分配的空间都记录在**memblock.reserved.regions[]**这个数组中，同样memblock.reserved.cnt是这个array的有效项数。这些entry同样都是memblock_region的base排序的（从小到大）,memblock.reserved.regions[memblock.reserved.cnt]

也就是说从memblock.memory.regions[]所表示的memory中去除memblock.reserved.regions[]记录的reserved空间就是可以用allocate的free memory!

dump memblock status info before mem_init()

MEMBLOCK configuration:

memory size = 0x3fc00000 reserved size = 0xfe0d3b

memory.cnt = 0x2

memory[0x0][0x0000000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

reserved.cnt = 0x18

reserved[0x0][0x00000000003000-0x00000000007fff], 0x5000 bytes flags: 0x0

reserved[0x1][0x00000000008300-0x000000006a68f3], 0x69e5f4 bytes flags: 0x0

reserved[0x2][0x00000000f00000-0x00000000f11fff], 0x12000 bytes flags: 0x0

reserved[0x3][0x0000002eed1000-0x0000002efa8fff], 0xd8000 bytes flags: 0x0

reserved[0x4][0x0000002efabd68-0x0000002f7ebfff], 0x840298 bytes flags: 0x0

reserved[0x5][0x0000002f7ec9c0-0x0000002f7eca03], 0x44 bytes flags: 0x0

reserved[0x6][0x0000002f7eca40-0x0000002f7eca83], 0x44 bytes flags: 0x0

reserved[0x7][0x0000002f7ecac0-0x0000002f7ecb37], 0x78 bytes flags: 0x0

reserved[0x8][0x0000002f7ecb40-0x0000002f7ecb47], 0x8 bytes flags: 0x0 reserved[0x9][0x0000002f7ecb80-0x0000002f7ecb87], 0x8 bytes flags: 0x0

reserved[0xa][0x0000002f7ecbc0-0x0000002f7ecbc3], 0x4 bytes flags: 0x0

reserved[0xb][0x0000002f7ecc00-0x0000002f7ecc75], 0x76 bytes flags: 0x0

reserved[0xc][0x0000002f7ecc80-0x0000002f7eccf5], 0x76 bytes flags: 0x0

reserved[0xd][0x0000002f7ecd00-0x0000002f7ecd75], 0x76 bytes flags: 0x0

reserved[0xe][0x0000002f7ecd80-0x0000002f7ecd83], 0x4 bytes flags: 0x0

reserved[0xf][0x0000002f7ecdb0-0x0000002f7ecec3], 0x114 bytes flags: 0x0

reserved[0x10][0x0000002f7ecec8-0x0000002f7ecee2], 0x1b bytes flags: 0x0

reserved[0x11][0x0000002f7ecee4-0x0000002f7ecefe], 0x1b bytes flags: 0x0

reserved[0x12][0x0000002f7ecf00-0x0000002f7ecf03], 0x4 bytes flags: 0x0

reserved[0x13][0x0000002f7ecf08-0x0000002f7ecf22], 0x1b bytes flags: 0x0

reserved[0x14][0x0000002f7ecf24-0x0000002f7ecf3e], 0x1b bytes flags: 0x0

reserved[0x15][0x0000002f7ecf40-0x0000002f7ecf43], 0x4 bytes flags: 0x0

reserved[0x16][0x0000002f7ecf50-0x0000002f7ecf6c], 0x1d bytes flags: 0x0

reserved[0x17][0x0000002f7ecf70-0x0000002f7fffff], 0x13090 bytes flags: 0x0

----

in arch/arm/mm/init.c

```
/*
 * mem_init() marks the free areas in the mem_map and tells us how much
 * memory is free.  This is done after various parts of the system have
 * claimed their memory after the kernel image.
 */
void __init mem_init(void)
{
#ifdef CONFIG_HAVE_TCM
	/* These pointers are filled in on TCM detection */
	extern u32 dtcm_end;
```

```
	extern u32 itcm_end;
#endif

	set_max_mapnr(pfn_to_page(max_pfn) - mem_map);                    ①

	/* this will put all unused low memory onto the freelists */
	free_unused_memmap();                                      ②
	free_all_bootmem();                                      ③

#ifdef CONFIG_SA1111
	/* now that our DMA memory is actually so designated, we can free it */
	free_reserved_area(__va(PHYS_OFFSET), swapper_pg_dir, -1, NULL);
#endif

	free_highpages();                                      ④

	mem_init_print_info(NULL);                                      ⑤

#define MLK(b, t) b, t, ((t) - (b)) >> 10                          ⑥
#define MLM(b, t) b, t, ((t) - (b)) >> 20
#define MLK_ROUNDUP(b, t) b, t, DIV_ROUND_UP(((t) - (b)), SZ_1K)

	printk(KERN_NOTICE "Virtual kernel memory layout:\n"
			"    vector  : 0x%08lx - 0x%08lx   (%4ld kB)\n"
#ifdef CONFIG_HAVE_TCM
```

```
                    "    DTCM    : 0x%08lx - 0x%08lx   (%4ld kB)\n"
                    "    ITCM    : 0x%08lx - 0x%08lx   (%4ld kB)\n"
#endif
                    "    fixmap  : 0x%08lx - 0x%08lx   (%4ld kB)\n"
                    "    vmalloc : 0x%08lx - 0x%08lx   (%4ld MB)\n"
                    "    lowmem  : 0x%08lx - 0x%08lx   (%4ld MB)\n"
#ifdef CONFIG_HIGHMEM
                    "    pkmap   : 0x%08lx - 0x%08lx   (%4ld MB)\n"
#endif
#ifdef CONFIG_MODULES
                    "    modules : 0x%08lx - 0x%08lx   (%4ld MB)\n"
#endif
                    "      .text : 0x%p" " - 0x%p" "   (%4td kB)\n"
                    "      .init : 0x%p" " - 0x%p" "   (%4td kB)\n"
                    "      .data : 0x%p" " - 0x%p" "   (%4td kB)\n"
                    "       .bss : 0x%p" " - 0x%p" "   (%4td kB)\n",

                    MLK(UL(CONFIG_VECTORS_BASE), UL(CONFIG_VECTORS_BASE) +
                          (PAGE_SIZE)),
#ifdef CONFIG_HAVE_TCM
                    MLK(DTCM_OFFSET, (unsigned long) dtcm_end),
                    MLK(ITCM_OFFSET, (unsigned long) itcm_end),
#endif
                    MLK(FIXADDR_START, FIXADDR_TOP),
                    MLM(VMALLOC_START, VMALLOC_END),
```

```
			MLM(PAGE_OFFSET, (unsigned long)high_memory),
#ifdef CONFIG_HIGHMEM
			MLM(PKMAP_BASE, (PKMAP_BASE) + (LAST_PKMAP) *
				(PAGE_SIZE)),
#endif
#ifdef CONFIG_MODULES
			MLM(MODULES_VADDR, MODULES_END),
#endif

			MLK_ROUNDUP(_text, _etext),
			MLK_ROUNDUP(__init_begin, __init_end),
			MLK_ROUNDUP(_sdata, _edata),
			MLK_ROUNDUP(__bss_start, __bss_stop));

#undef MLK
#undef MLM
#undef MLK_ROUNDUP

	/*
	 * Check boundaries twice: Some fundamental inconsistencies can
	 * be detected at build time already.
	 */
#ifdef CONFIG_MMU
	BUILD_BUG_ON(TASK_SIZE				> MODULES_VADDR);
	BUG_ON(TASK_SIZE				> MODULES_VADDR);
```

```
#endif

#ifdef CONFIG_HIGHMEM
        BUILD_BUG_ON(PKMAP_BASE + LAST_PKMAP * PAGE_SIZE > PAGE_OFFSET);
        BUG_ON(PKMAP_BASE + LAST_PKMAP * PAGE_SIZE    > PAGE_OFFSET);
#endif

if (PAGE_SIZE >= 16384 && get_num_physpages() <= 128) {
                extern int sysctl_overcommit_memory;
                /*
                 * On a machine this small we won't get
                 * anywhere without overcommit, so turn
                 * it on by default.
                 */
                sysctl_overcommit_memory = OVERCOMMIT_ALWAYS;
        }
}
```

①

设置mem_map[] array的最大有效index，即max_mapnr。

```
unsigned long max_mapnr;
```

max_pfn是最大的page frame,而pfn_to_page(max_pfn)转换page frame到对应的struct page。

struct page *mem_map;

one page frame, one struct page.

mem_map[]是struct page的array。

438e08f1-r0/image/boot$ nm vmlinux-3.18.7-yocto-standard | grep " mem_map"

c0667080 B mem_map

mem_map[0]   -->   physical page 0

mem_map[1]   --> physical page 1

......

mem_map[262143] --> physical page 262143 (Gr2有1G memory)

max_mapnr = 262144；

index of mem_map[] < max_mapnr

②

```
/*
 * The mem_map array can get very big.  Free the unused area of the memory map.
 */
static void __init free_unused_memmap(void)
```

```
{
        unsigned long start, prev_end = 0;
        struct memblock_region *reg;

        /*
         * This relies on each bank being in address order.
         * The banks are sorted previously in bootmem_init().
         */
        for_each_memblock(memory, reg) {                        (A)
                start = memblock_region_memory_base_pfn(reg);          (A.1)

#ifdef CONFIG_SPARSEMEM
                /*
                 * Take care not to free memmap entries that don't exist
                 * due to SPARSEMEM sections which aren't present.
                 */
                start = min(start,
                                ALIGN(prev_end, PAGES_PER_SECTION));
#else
                /*
                 * Align down here since the VM subsystem insists that the
                 * memmap entries are valid from the bank start aligned to
                 * MAX_ORDER_NR_PAGES.
                 */
                start = round_down(start, MAX_ORDER_NR_PAGES);         (B)
```

```
#endif
		/*
		 * If we had a previous bank, and there is a space
		 * between the current bank and the previous, free it.
		 */
		if (prev_end && prev_end < start)			(C)
			free_memmap(prev_end, start);

		/*
		 * Align up here since the VM subsystem insists that the
		 * memmap entries are valid from the bank end aligned to
		 * MAX_ORDER_NR_PAGES.
		 */
		prev_end = ALIGN(memblock_region_memory_end_pfn(reg),	(D)
				 MAX_ORDER_NR_PAGES);
	}

#ifdef CONFIG_SPARSEMEM
	if (!IS_ALIGNED(prev_end, PAGES_PER_SECTION))
		free_memmap(prev_end,
			    ALIGN(prev_end, PAGES_PER_SECTION));
#endif
}
```

(A)

对memblock.memory.regions[] array进行enumerate.

memory size = 0x3fc00000 reserved size = 0xfe0d3b

memory.cnt = 0x2

memory[0x0][0x0000000000000000-0x00000005ffffff], 0x6000000 bytes flags: 0x0

memory[0x1][0x00000006400000-0x0000003fffffff], 0x39c00000 bytes flags: 0x0

memblock.memory.regions[0].base = 0

memblock.memory.regions[0].size = 0x6000000 (96M)

memblock.memory.regions[0].base = 0x6400000

memblock.memory.regions[0].size = 0x39c00000 (924M)

当中没挖去的4M给Secure core R4用的。

(A.1)

使得base向上取整到page边界。比如base = 5000,向上取整后就是8192 (0x2000)

对memblock.memory.regions[0].base = 0而言，start = 0

对memblock.memory.regions[1].base = 0x6400000 (100M)而言，start = 0x6400000(本来就在page alignment).

(B)

CONFIG_FORCE_MAX_ZONEORDER=16

in include/linux/mmzone.h

```
/* Free memory management - zoned buddy allocator.  */
#ifndef CONFIG_FORCE_MAX_ZONEORDER
#define MAX_ORDER 11
#else
#define MAX_ORDER CONFIG_FORCE_MAX_ZONEORDER
#endif
#define MAX_ORDER_NR_PAGES (1 << (MAX_ORDER - 1))
```

MAX_ORDER_NR_PAGES为15, 即2 ^ 15 pages = 32K pages = **128M**

再向下取整到32K page边界，即128M的边界。

对start = 0而言，start取整后还是0

对start = 0x6400000而言，100M < 128M ==> start = 0

??? 是否代表在大可分配连续physical memory为128M呢 ???

??? Linux kernel可以分配的最大连续physical memory是多大 ???

(C)

在enumerate memblock.memory.regions[0]时，prev_end = 0

在enumerate memblock.memory.regions[1]时，prev_end = 0

所以条件都不满足。也就是free_unused_memmap()什么都没做，等于没调用一样！

(D)

对region上界(base + size)在128M上向下取整。

由于memblock.memory.regions[0].base + memblock.memory.regions[0].size = 96M < 128M

==>

prev_end = 0

③

真正完成从memblock到buddy allocator的交接在free_all_bootmem() function中。

in mm/nobootmem.c

```
/**
 * free_all_bootmem - release free pages to the buddy allocator
 *
 * Returns the number of pages actually released.
 */
unsigned long __init free_all_bootmem(void)
{
        unsigned long pages;

        reset_all_zones_managed_pages();

        /*
         * We need to use NUMA_NO_NODE instead of NODE_DATA(0)->node_id
```

```
         *  because in some case like Node0 doesn't have RAM installed
         *  low ram will be on Node1
*/
        pages = free_low_memory_core_early();
        totalram_pages += pages;

        return pages;
}
```

in mm/nobootmem.c

```
static unsigned long __init free_low_memory_core_early(void)
{
        unsigned long count = 0;
        phys_addr_t start, end;
        u64 i;

        memblock_clear_hotplug(0, -1);

        for_each_free_mem_range(i, NUMA_NO_NODE, &start, &end, NULL)   (A)
                count += __free_memory_core(start, end);                (B)

#ifdef CONFIG_ARCH_DISCARD_MEMBLOCK
        {
                phys_addr_t size;
```

```
		/* Free memblock.reserved array if it was allocated */
		size = get_allocated_memblock_reserved_regions_info(&start);
		if (size)
			count += __free_memory_core(start, start + size);

		/* Free memblock.memory array if it was allocated */
		size = get_allocated_memblock_memory_regions_info(&start);
		if (size)
			count += __free_memory_core(start, start + size);
	}
#endif

	return count;
}
```

(A)

(B)

```
/**
 * for_each_free_mem_range - iterate through free memblock areas
 * @i: u64 used as loop variable
 * @nid: node selector, %NUMA_NO_NODE for all nodes
 * @p_start: ptr to phys_addr_t for start address of the range, can be %NULL
 * @p_end: ptr to phys_addr_t for end address of the range, can be %NULL
```

```
 * @p_nid: ptr to int for nid of the range, can be %NULL
 *
 * Walks over free (memory && !reserved) areas of memblock.  Available as
 * soon as memblock is initialized.
 */
#define for_each_free_mem_range(i, nid, p_start, p_end, p_nid)         \
        for_each_mem_range(i, &memblock.memory, &memblock.reserved,\
                           nid, p_start, p_end, p_nid)
```

==>

```
/**
 * for_each_mem_range - iterate through memblock areas from type_a and not
 * included in type_b. Or just type_a if type_b is NULL.
 * @i: u64 used as loop variable
 * @type_a: ptr to memblock_type to iterate
 * @type_b: ptr to memblock_type which excludes from the iteration
 * @nid: node selector, %NUMA_NO_NODE for all nodes
 * @p_start: ptr to phys_addr_t for start address of the range, can be %NULL
 * @p_end: ptr to phys_addr_t for end address of the range, can be %NULL
 * @p_nid: ptr to int for nid of the range, can be %NULL
 */
#define for_each_mem_range(i, type_a, type_b, nid,                     \
                           p_start, p_end, p_nid)                 \
```

```
	for (i = 0, __next_mem_range(&i, nid, type_a, type_b,		\
				     p_start, p_end, p_nid);		\
	     i != (u64)ULLONG_MAX;					\
	     __next_mem_range(&i, nid, type_a, type_b,			\
			      p_start, p_end, p_nid))
```

in mm/memblock.c

```
/**
 * __next_mem_range_rev - generic next function for for_each_*_range_rev()
 *
 * Finds the next range from type_a which is not marked as unsuitable
 * in type_b.
 *
 * @idx: pointer to u64 loop variable
 * @nid: nid: node selector, %NUMA_NO_NODE for all nodes
 * @type_a: pointer to memblock_type from where the range is taken
 * @type_b: pointer to memblock_type which excludes memory from being taken
 * @out_start: ptr to phys_addr_t for start address of the range, can be %NULL
 * @out_end: ptr to phys_addr_t for end address of the range, can be %NULL
 * @out_nid: ptr to int for nid of the range, can be %NULL
 *
 * Reverse of __next_mem_range().
 */
```

```
void __init_memblock __next_mem_range_rev(u64 *idx, int nid,
                                    struct memblock_type *type_a,
                                    struct memblock_type *type_b,
                                    phys_addr_t *out_start,
                                    phys_addr_t *out_end, int *out_nid)
```

???

④

```
static void __init free_highpages(void)
{
#ifdef CONFIG_HIGHMEM
        unsigned long max_low = max_low_pfn;
        struct memblock_region *mem, *res;

        /* set highmem page free */
        for_each_memblock(memory, mem) {                                    (A)
                unsigned long start = memblock_region_memory_base_pfn(mem);
                unsigned long end = memblock_region_memory_end_pfn(mem);

                /* Ignore complete lowmem entries */
                if (end <= max_low)
                        continue;
```

```
/* Truncate partial highmem entries */

if (start < max_low)

        start = max_low;
                                                                (B)


/* Find and exclude any reserved regions */

for_each_memblock(reserved, res) {                                          (C)

        unsigned long res_start, res_end;


        res_start = memblock_region_reserved_base_pfn(res);             (D)

        res_end = memblock_region_reserved_end_pfn(res);            (E)


        if (res_end < start)                        (F)

                continue;

        if (res_start < start)                                              (G)


                res_start = start;

        if (res_start > end)                                                (H)

                res_start = end;

        if (res_end > end)
                                                                (I)

                res_end = end;

        if (res_start != start)

                free_area_high(start, res_start);

        start = res_end;

        if (start == end)
```

```
                    break;
            }

            /* And now free anything which remains */
            if (start < end)
                free_area_high(start, end);
        }
#endif
}
```

(A)

对memblock.memory.regions[] array enumerate。

memblock.memory.regions[0]不满足条件。

(B)

start = memblock.memory.regions[1].base = 100M < max_low,so start = max_low

(C)

在memblock.memory.regions[1]中的memory从max_low到(memblock.memory.regions[1].base + memblock.memory.regions[1].size)属于high memory，但要去出reserved regions,即memblock.reserved.regions[] array中的space。

reserved.cnt = 0x18

reserved[0x0][0x0000000003000-0x000000000007fff], 0x5000 bytes flags: 0x0 reserved[0x1][0x0000000008300-0x000000006a68f3], 0x69e5f4 bytes flags: 0x0

reserved[0x2][0x00000000f00000-0x00000000f11fff], 0x12000 bytes flags: 0x0

下面的reserved space位于high memory

reserved[0x3][0x0000002eed1000-0x0000002efa8fff], 0xd8000 bytes flags: 0x0

reserved[0x4][0x0000002efabd68-0x0000002f7ebfff], 0x840298 bytes flags: 0x0

reserved[0x5][0x0000002f7ec9c0-0x0000002f7eca03], 0x44 bytes flags: 0x0

reserved[0x6][0x0000002f7eca40-0x0000002f7eca83], 0x44 bytes flags: 0x0

reserved[0x7][0x0000002f7ecac0-0x0000002f7ecb37], 0x78 bytes flags: 0x0

reserved[0x8][0x0000002f7ecb40-0x0000002f7ecb47], 0x8 bytes flags: 0x0

reserved[0x9][0x0000002f7ecb80-0x0000002f7ecb87], 0x8 bytes flags: 0x0

reserved[0xa][0x0000002f7ecbc0-0x0000002f7ecbc3], 0x4 bytes flags: 0x0

reserved[0xb][0x0000002f7ecc00-0x0000002f7ecc75], 0x76 bytes flags: 0x0 reserved[0xc][0x0000002f7ecc80-0x0000002f7eccf5], 0x76 bytes flags: 0x0

reserved[0xd][0x0000002f7ecd00-0x0000002f7ecd75], 0x76 bytes flags: 0x0

reserved[0xe][0x0000002f7ecd80-0x0000002f7ecd83], 0x4 bytes flags: 0x0

reserved[0xf][0x0000002f7ecdb0-0x0000002f7ecec3], 0x114 bytes flags: 0x0

reserved[0x10][0x0000002f7ecec8-0x0000002f7ecee2], 0x1b bytes flags: 0x0

reserved[0x11][0x0000002f7ecee4-0x0000002f7ecefe], 0x1b bytes flags: 0x0

reserved[0x12][0x0000002f7ecf00-0x0000002f7ecf03], 0x4 bytes flags: 0x0

reserved[0x13][0x0000002f7ecf08-0x0000002f7ecf22], 0x1b bytes flags: 0x0

reserved[0x14][0x0000002f7ecf24-0x0000002f7ecf3e], 0x1b bytes flags: 0x0

reserved[0x15][0x0000002f7ecf40-0x0000002f7ecf43], 0x4 bytes flags: 0x0

reserved[0x16][0x0000002f7ecf50-0x0000002f7ecf6c], 0x1d bytes flags: 0x0

reserved[0x17][0x0000002f7ecf70-0x0000002f7fffff], 0x13090 bytes flags: 0x0

对位于high memory空间的reserved space进行enumerate。

(D)

(E)

res_start, res_end都对齐在page边界上，并且包括了reserved的region。

(F)

表示该reserved region不再heigh memory之内

(G)

表示该reserved region的下边界（低地址的一端）落在high memory的外面（即在normal memory）

(H)

表示该reserved region的下边界（低地址的一端）超出了当前memory bank的end边界（reserved region跨memory bank，应该不大可能）

也就是reserved region根本不在当前memory bank之内

(I)

表示该reserved region的上边界（高地址的一端）超出了当前memory bank的end边界

code看上去由很多判断，实际在G2 LSP上没那么复杂，就是从memblock.memory.regions[1]代表的内存中从high memory的边界开始，凡是没有reserved region的身影的page都调用free_area_high().

```
#ifdef CONFIG_HIGHMEM

static inline void free_area_high(unsigned long pfn, unsigned long end)

{
```

```
	for (; pfn < end; pfn++)
		free_highmem_page(pfn_to_page(pfn));
}
#endif

#ifdef	CONFIG_HIGHMEM
void free_highmem_page(struct page *page)
{
	__free_reserved_page(page);
	totalram_pages++;
	page_zone(page)->managed_pages++;
	totalhigh_pages++;
}
#endif
```

⑤

该函数print如下信息：

Memory: 1028212K/1044480K available (4550K kernel code, 217K rwdata, 1352K rodata, 208K init, 448K bss, 16268K reserved, 270336K highmem)

```
void __init mem_init_print_info(const char *str)
{
	unsigned long physpages, codesize, datasize, rosize, bss_size;
	unsigned long init_code_size, init_data_size;
```

```
        physpages = get_num_physpages();

        codesize = _etext - _stext;

        datasize = _edata - _sdata;

        rosize = __end_rodata - __start_rodata;

        bss_size = __bss_stop - __bss_start;

        init_data_size = __init_end - __init_begin;

        init_code_size = _einittext - _sinittext;


        /*

         * Detect special cases and adjust section sizes accordingly:

         * 1) .init.* may be embedded into .data sections

         * 2) .init.text.* may be out of [__init_begin, __init_end],

         *    please refer to arch/tile/kernel/vmlinux.lds.S.

         * 3) .rodata.* may be embedded into .text or .data sections.

         */

#define adj_init_size(start, end, size, pos, adj) \

        do { \

                if (start <= pos && pos < end && size > adj) \

                        size -= adj; \

        } while (0)


        adj_init_size(__init_begin, __init_end, init_data_size,                    (A)

                      _sinittext, init_code_size);

        adj_init_size(_stext, _etext, codesize, _sinittext, init_code_size);  (B)
```

```
        adj_init_size(_sdata, _edata, datasize, __init_begin, init_data_size);(C)

        adj_init_size(_stext, _etext, codesize, __start_rodata, rosize);          (D)

        adj_init_size(_sdata, _edata, datasize, __start_rodata, rosize);          (E)


#undef    adj_init_size


        printk("Memory: %luK/%luK available "

               "(%luK kernel code, %luK rwdata, %luK rodata, "

               "%luK init, %luK bss, %luK reserved"

#ifdef     CONFIG_HIGHMEM

               ", %luK highmem"

#endif

               "%s%s)\n",

               nr_free_pages() << (PAGE_SHIFT-10), physpages << (PAGE_SHIFT-10),

               codesize >> 10, datasize >> 10, rosize >> 10,

               (init_data_size + init_code_size) >> 10, bss_size >> 10,

               (physpages - totalram_pages) << (PAGE_SHIFT-10),

#ifdef     CONFIG_HIGHMEM

               totalhigh_pages << (PAGE_SHIFT-10),

#endif

               str ? ", " : "", str ? str : "");

}


(A)

if(__init_begin < _sinittext && _sinittext < __init_end && init_data_size > init_code_size)
```

init_data_size -= init_code_size;

表示_sinittext是被包含在(__init_begin, __init_end)之间的，自然init_data_size应该减去init_code_size的大小（否则就重复计算了）

(B)

如果_sinittext被包含在_stext里，则也要调整大小。

(C)

如果__init_begin被包含在_sdata里，则也要调整大小。

(D)

如果__start_rodata被包含在_stext里，则也要调整大小。

(E)

如果__start_rodata被包含在_sdata里，则也要调整大小。

physpages ???

totalram_pages ???

totalhigh_pages ???

⑥

in Gr2 LSP, memory info as follow

Memory: 1028212K/1044480K available (4550K kernel code, 217K rwdata, 1352K rodata, 208K init, 448K bss, 16268K reserved, 270336K highmem)

Virtual kernel memory layout:

```
  vector  : 0xffff0000 - 0xffff1000   (   4 kB)
  fixmap  : 0xffc00000 - 0xffe00000   (2048 kB)
  vmalloc : 0xf0000000 - 0xff000000   ( 240 MB)
  lowmem  : 0xc0000000 - 0xef800000   ( 760 MB)
  pkmap   : 0xbfe00000 - 0xc0000000   (   2 MB)
  modules : 0xbf000000 - 0xbfe00000   (  14 MB)
    .text : 0xc0008000 - 0xc05cbdd8   (5904 kB)
    .init : 0xc05cc000 - 0xc0600000   ( 208 kB)
    .data : 0xc0600000 - 0xc0636518   ( 218 kB)
```

.bss : 0xc0636518 - 0xc06a68f4   (449 kB)