4 Compilation with GNU Tools

내용

- C 프로그램 컴파일 과정
- ELF 파일 구조
- Memory Map - BeagleBone의 SDRAM
- Little Endian vs Big Endian
- C Start Up Code - 파일 crt0.S
- Link Script - 파일 link.lds
- 컴파일 보기 - args.c 및 Makefile

C 프로그램 컴파일 과정

순서	단계 이름	명령어	입력 파일	출력 파일
1	Preprocess	gcc, cpp	C 혹은 assembler 파일 (.c 혹은 .S)	C 혹은 assembler 파일 (.i 혹은 .s)
2	Compile	gcc	C 파일 (.i)	Assembler 파일 (.s)
3	Assemble	gcc, gas	Assembler 파일 (.s)	ELF(Executable Linking Format) 형식의 object 파일 (.o)
4	Link	gcc, ld	ELF 형식의 다수의 object 파일 (.o) 및 linker script	ELF 형식의 executable 파일

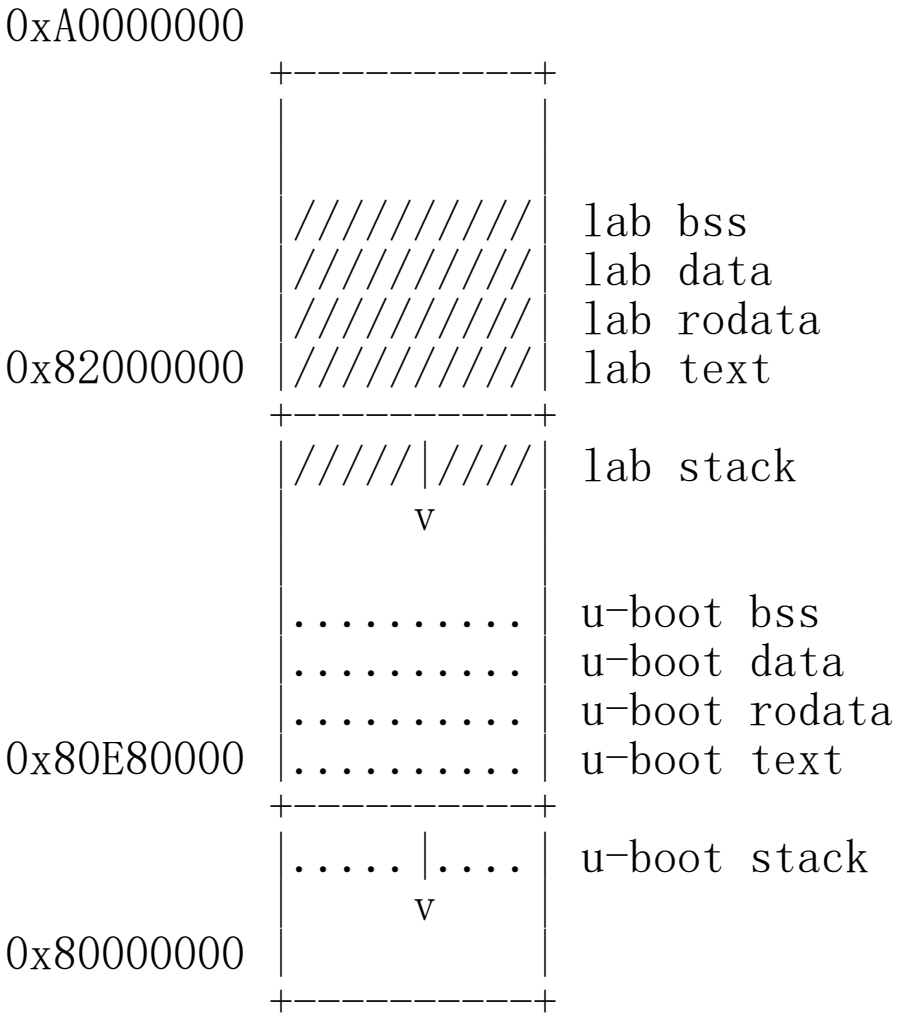
ELF 파일 구조

- Object/shared library/executable/core dump 파일을 표현하는 형식으로 아래와 같은 요소로 구성됨
  1. ELF header: ELF 파일의 전반적인 정보
  2. Program header들: 실행에 필요한 segment들에 대한 정보
  3. Section들: 실제 section들의 내용
  4. Section header들: section에 대한 정보
- 하나의 segment는 여러개의 section을 가지고 있음
  - 0번 segment: .text, .rodata, .data, .bss 포함
- 어떤 section은 실행과 관련 없기 때문에 어떤 segment에도 포함되지 않음
  - .comment, .ARM.attributes, .debug_*, .symtab 등

ELF 파일 관련 주요 명령어

명령어	설명
`$ readelf -a ELF_파일`	ELF 파일의 주요 정보를 출력 (ELF header, section header, program header 등)
`$ size -A ELF_파일`	ELF 파일의 각 section의 가상 메모리 시작 주소 및 크기를 출력 (주: segment에 포함되지 않는 section의 주소 정보는 없음)
`$ objdump -DSx ELF_파일`	ELF 파일의 각 section을 disassemble하여 가상 메모리 주소와 instruction을 출력

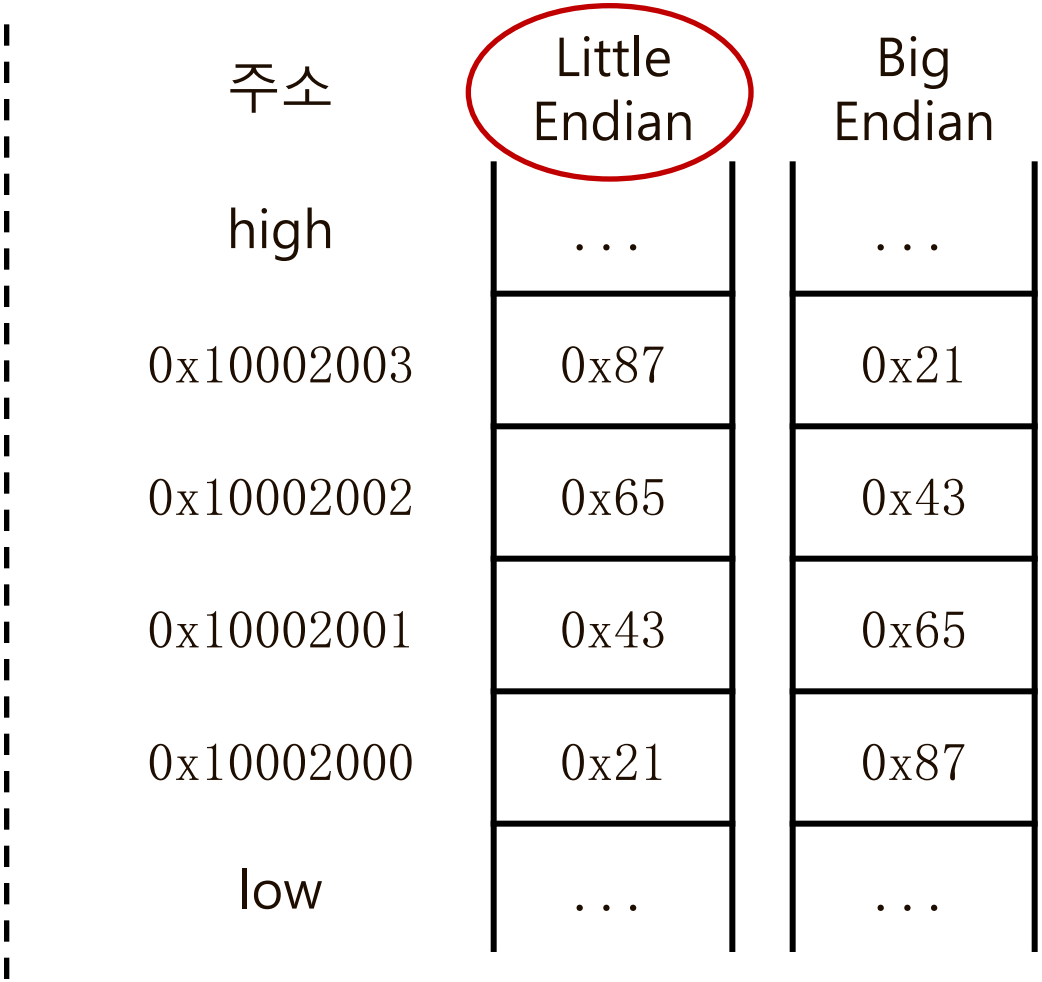
Memory Map – BeagleBone의 SDRAM



Little Endian vs Big Endian

```
unsigned int *p =
   0x10002000;

*p = 0x87654321;
```



C Start Up Code - 파일 start.S

- C 프로그램을 시작시키는 프로그램
- 우리의 실습 프로그래밍의 경우 SDRAM의 0x82000000 위치에 저장됨
- U-Boot에서 "go 82000000 arg1 arg2 ..."를 수행하면 시작됨
- U-Boot 명령어 "go"는 argc와 argv를 설정한 후 함수 main(argc, argv)을 부름
- 주요 기능
  - Initialization of the bss section
  - call main(argc, argv)
  - return to boot loader

파일 start.S - 1/2

```
.globl _start
_start:
    push {lr}                   // save lr
    str  sp, _saved_sp          // save u-boot sp
    ldr  sp, _svc_stack         // set svc stack

    ldr  r4, _bss_start         // r4=bss_start
    ldr  r5, _bss_end           // r5=bss_end
    cmp  r4, r5                 // if r4==r5
    beq  .L1                    // then goto .L1
    mov  r6, #0                 // r6=0
.L0:
    str  r6, [r4]               // *r4=r6
    add  r4, r4, #4                     // r4=r4+4
    cmp  r4, r5                 // if r4!=r5
    bne  .L0                    // then goto .L0
.L1:
```

파일 start.S - 2/2

```
    bl    main                          // main(argc, argv);

    ldr   sp, _saved_sp                 // restore u-boot sp
    pop   {pc}                          // restore pc

.globl _bss_start
_bss_start:
    .word __bss_start

.globl _bss_end
_bss_end:
    .word _end

_saved_sp:
    .word 0x00000000

_svc_stack:
    .word 0x82000000-0x00000000
```

Link Script - 파일 link.lds

- Link 단계에서 다수의 ELF 형식의 object 파일을 편집하여 하나의 executable 파일을 생성하는 데 필요한 편집 정보를 가지고 있는 파일
- 명령어 ld의 인수로 link script 파일을 줄 수 있음
  -T link.lds 혹은 -script=link.lds
- 우리의 실습 프로그래밍에서 사용되는 linker script 내용
  - 프로그램은 _start 번지에서 수행을 시작함
  - 첫 section의 시작 주소는 0x82000000
  - Section .text, .rodata, .data, .bss를 편집 방법에 대한 기술
- Linker script 내에서는 변수(예: __bss_start, _end)를 사용하여 프로그램 수행 중 어떤 section의 시작 주소 등의 정보를 알아낼 수 있음

파일 link.lds

```
OUTPUT_FORMAT("elf32-littlearm", "elf32-littlearm", "elf32-littlearm")
OUTPUT_ARCH(arm)
ENTRY(_start)
SECTIONS
{
  . = 0x82000000;
  . = ALIGN(4);
  .text : { *(.text) }

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

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

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

컴파일 보기 - args.c

```
#include <uart.h>

int main(int argc, char *argv[])
{
  int k;

  UART_printf("argc=%d\n", argc);
  for (k = 0; k < argc; k++) {
    UART_printf("argv[%d]=%s\n", k, argv[k]);
  }

  return 7; // return 7 to U-boot!
}
```

컴파일 보기 - Makefile - 1/2

```
# ========================================================================

CC = arm-eabi-gcc
AS = arm-eabi-as
LD = arm-eabi-ld
OBJCOPY = arm-eabi-objcopy
OBJDUMP = arm-eabi-objdump

# ========================================================================

CFLAGS = -g -Wall -marm -mcpu=cortex-a8 -mfpu=neon -mfloat-abi=softfp -
    mabi=aapcs-linux -I. -I../libuart
ASFLAGS = -g -mcpu=cortex-a8 -mfpu=neon
LDFLAGS = -L ../libuart -L $(shell dirname `$(CC) -print-libgcc-file-name`)
LIBRARIES = -luart -lgcc
LINKSCRIPT = link.lds

# ========================================================================
```

컴파일 보기 - Makefile - 2/2

```
# ============================================================

all:    args.elf args.bin args.dis

# ============================================================

args.elf:       $(LINKSCRIPT) start.o args.o
        $(LD) -T $(LINKSCRIPT) start.o args.o $(LIBRARIES) $(LDFLAGS) -o
    args.elf

args.bin:       args.elf
        $(OBJCOPY) -O binary args.elf args.bin

args.dis:       args.elf
        $(OBJDUMP) -DSx args.elf > args.dis

# ============================================================
```

Makefile Macros

- `CC = arm-eabi-gcc`
  - GNU C compiler
- `AS = arm-eabi-as`
  - GNU assembler
- `LD = arm-eabi-ld`
  - GNU linker
- `OBJCOPY = arm-eabi-objcopy`
  - Copy and translate object files
- `OBJDUMP = arm-eabi-objdump`
  - Display object file information

Compilation Flags

- CFLAGS = -g -Wall -marm -mcpu=cortex-a8 -mfpu=neon -mfloat-abi=softfp -mabi=aapcs-linux -I. -I../libuart
- ASFLAGS = -g -mcpu=cortex-a8 -mfpu=neon
- LDFLAGS = -L ../libuart -L $(shell dirname `$(CC) -print-libgcc-file-name`)
- LIBRARIES = -luart -lgcc
- LINKSCRIPT = link.lds

Makefile Rule

```
args.elf: $(LINKSCRIPT) start.o args.o
        $(LD) -T $(LINKSCRIPT) start.o args.o
  $(LIBRARIES) $(LDFLAGS) -o args.elf
...
```

컴파일 과정

1. Preprocess
   - arm-eabi-gcc를 수행하여 .c 파일을 .i 파일로 변환
   - arm-eabi-gcc를 수행하여 .S 파일을 .s 파일로 변환
2. Compile
   - arm-eabi-gcc를 수행하여 .i 파일을 .o 파일로 변환 (옵션 CFLAGS 사용)
3. Assemble
   - arm-eabi-as를 수행하여 .s 파일을 .o 파일로 변환 (옵션 ASFLAGS 사용)
4. Link
   - arm-eabi-ld를 수행하여 다수의 .o 파일을 연결하여 하나의 .elf 파일로 변환 (옵션 LDFLAGS, , LIBRARIES 및 -T LINKSCRIPT 사용)
5. Generate binary
   - arm-eabi-objcopy를 수행하여 .elf 파일을 .bin 파일로 변환
6. Disassemble
   - arm-eabi-objdump를 수행하여 .elf 파일을 disassemble함 (옵션 -DSx 사용)

참고 문헌

1. TIS Committee, Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification Version 1.2, May 1995.
2. The Santa Cruz Operation, System V Application Binary Interface Edition 4.1 , March 1997.
3. Free Software Foundation, Inc., Using as The GNU Assembler (GNU Binutils) Version 2.24.0, 2013.
4. Free Software Foundation, Inc., The GNU Binary Utilities (GNU Binutils) Version 2.24.0, August 2014.
5. Free Software Foundation, Inc., The C Preprocessor For GCC Version 4.9.2, 2014.
6. Free Software Foundation, Inc., Using the GNU Compiler Collection For GCC Version 4.9.2, 2014.
7. Free Software Foundation, Inc., The GNU Linker ld (GNU Binutils) Version 2.24.0, 2013.
8. Free Software Foundation, Inc., The GNU C Library Reference Manual, 2014.