Real Time System

36848-01

Lecture 1

이덕우

소 개

- 수업
  - 월 : 14:00 ~ 15:50
  - 수 : 09:00 ~ 10:50
- 면담 / 질문
  - dwoolee@ {kmu.ac.kr 또는 gmail.com}
  - 053-580-5268
  - 이메일 선호
- 이론강의 + 실습

소 개 (계속)

- 첫 2주간 온라인 강의
  - 해당 수업 날짜에 녹화된 영상을 게시함
  - 해당 수업 날짜 : 3월 16일, 3월 18일, 3월 23일, 3월 25일
- 평가
  - 중간고사 + 기말고사 + 과제 + 출석
  - 과제는 mini project로 대체될 수 있음

VM WARE 우분투 설치

- 리눅스 실습을 위함
- 실습 컴퓨터에 우분투 설치하여 리눅스 실습이 가능하도록 할 것
- 우분투 설치에 대한 설명은 웹사이트 참고

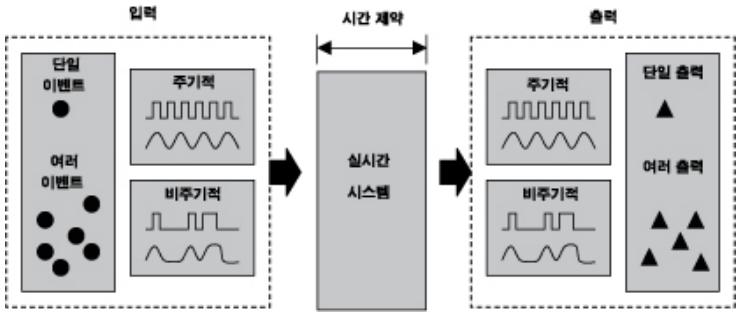
Embedded System

- 공장 자동화
- 방어 시스템
- 수송장치
- 항공장치
- 가정, 직장, 여가 등등.....

Embedded System

- 특정 기능을 수행하기 위해 하드웨어와 소프트웨어를 밀접하게 통합한 컴퓨팅 장치
- 큰 시스템 안에 내장(embedded) 되어 있는 또 하나의 시스템

Real Time System

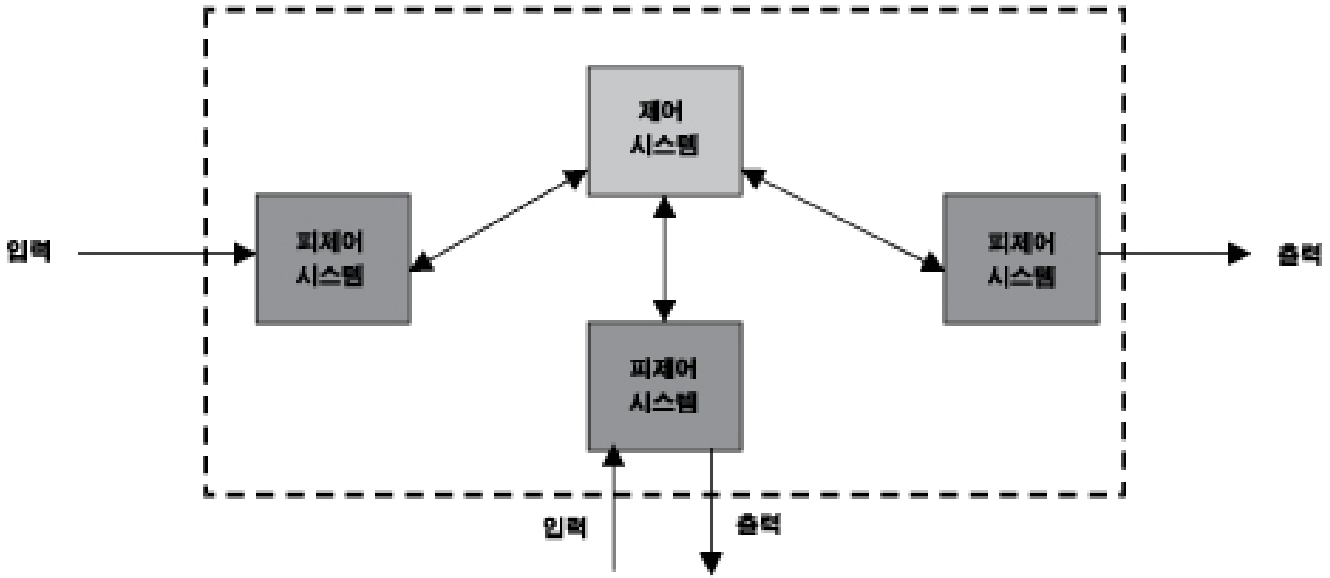


- 외부 이벤트에 대해 일정한 시간에 응답해야 하는 시스템
- 이벤트 발생 시점을 알아야 한다 → 적절한 시점에 적절한 처리를 할 수 있으며 주어진 시간 안에 필요한 결과를 출력할 수 있다.

Real Time Embedded System

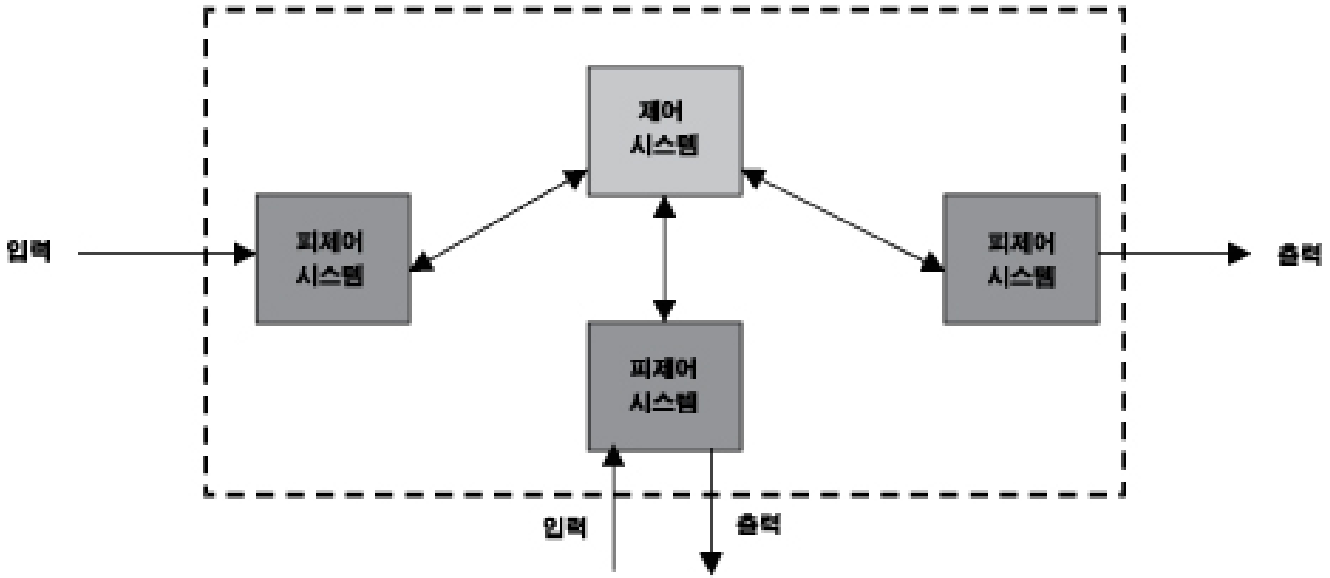
- 임베디드 시스템
- 실시간 시스템

Real Time System



- 외부 이벤트 항상 발생
- 이벤트는 시스템에 전달(입력)
- 시스템은 이벤트에 대한 응답을 보여줌 (출력)

Real Time System



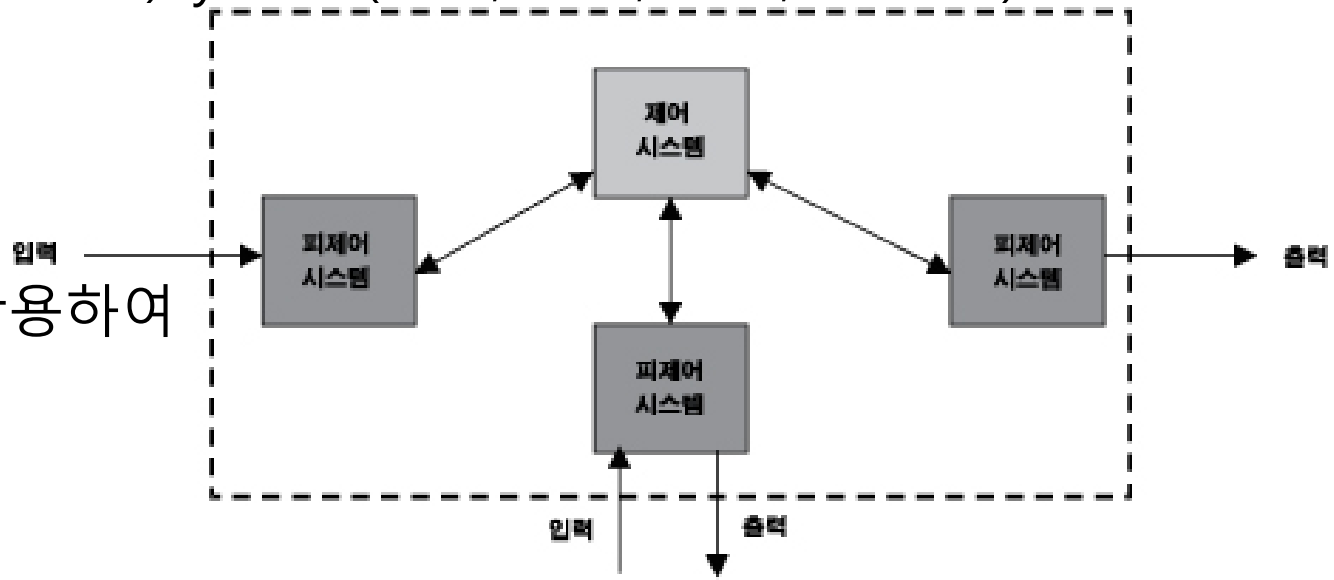
- 제어 시스템 + 피제어 시스템
- 주기적 시스템 : 제어 ←→ 피제어
- 비주기적 시스템 : 피제어 ←→ 제어

Real Time System

- 예시 (무기방어 시스템)
  - 시스템의 임무 : 다가오는 공격을 탐지하여 무력화 하고 아군의 구축함을 보호
  - 시스템의 기본 아이디어 : 미사일이 구축함에 도착하기 전에 폭파 시키는 것
  - 시스템의 구성 : 레이더(radar) 시스템, C&D (command and decision) 시스템, 무기발사 제어시스템 등

Real Time System

- 예시 (무기방어 시스템)
  - Radar System (탐지, 탐색 등)
  - C&D (Command and Decision)System (판단, 결정, 평가, 연산 등)
  - 무기발사 제어시스템
  - Q1 : 위 세가지 요소를 활용하여 시나리오 구성 (20mins)



Real Time System

- 예시 (무기방어 시스템)
  - Radar System : 잠재 목표물을 탐색하고 찾는다
  - 찾으면 C&D로 정보를 송신
  - C&D 는 Radar가 송신한 정보를 근거로 위협 수준 등을 평가, 판단 등
  - 무기발사 제어시스템 C&D 의 명령에 따라 작동

  - Q2 : 주기 시스템? 비주기 시스템? (20mins)

Real Time System

- 예시 (무기방어 시스템)
  - Radar vs C&D
  - C&D vs 무기발사제어 시스템
  - Radar vs 무기발사제어 시스템

Real Time System

- 예시 (DVD 재생기)
  - DVD 재생기
  - 리모콘

Real Time System

- 중요한 사항
  - 기능
  - 시간
  - 시간 정확성 (timing correctness)
  - 기능 정확성 (functional correctness)

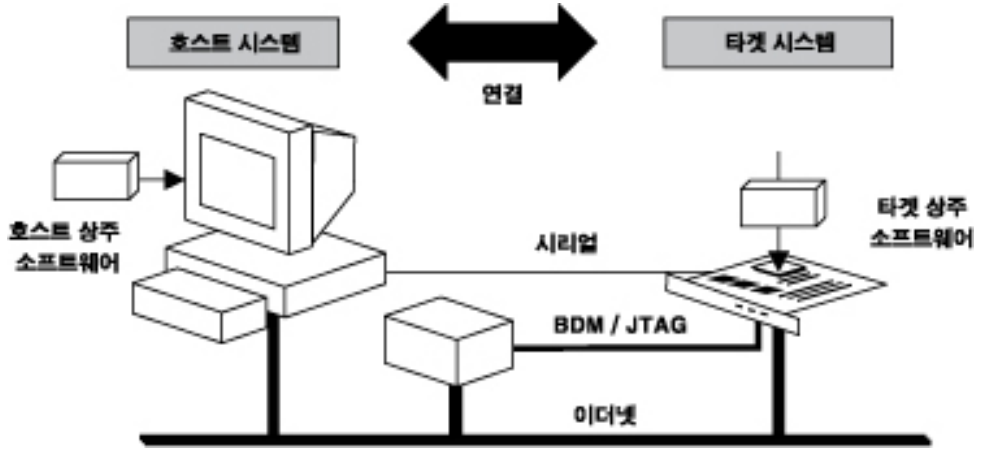
Real Time System

- Hard RTS
  - 무기, 항법 등
- Soft RTS
  - DVD 재생기 등

Real Time System

- 제품 시장의 주기는 약 6개월
- 응용분야의 다양성 (임베디드 시스템은 특정 용도를 위한 것임)
- 네트워크 기능은 필수
- 기능의 복잡성 증가 추세
- 기존의 임베디드 시스템이 새로운 응용 임베디드 시스템 창출

Host / Target System



- Host System (or Host PC) : 프로그램을 개발하는 PC
- Target System (or Target) : 프로그램 결과물을 실행할 환경 (또는 시스템)

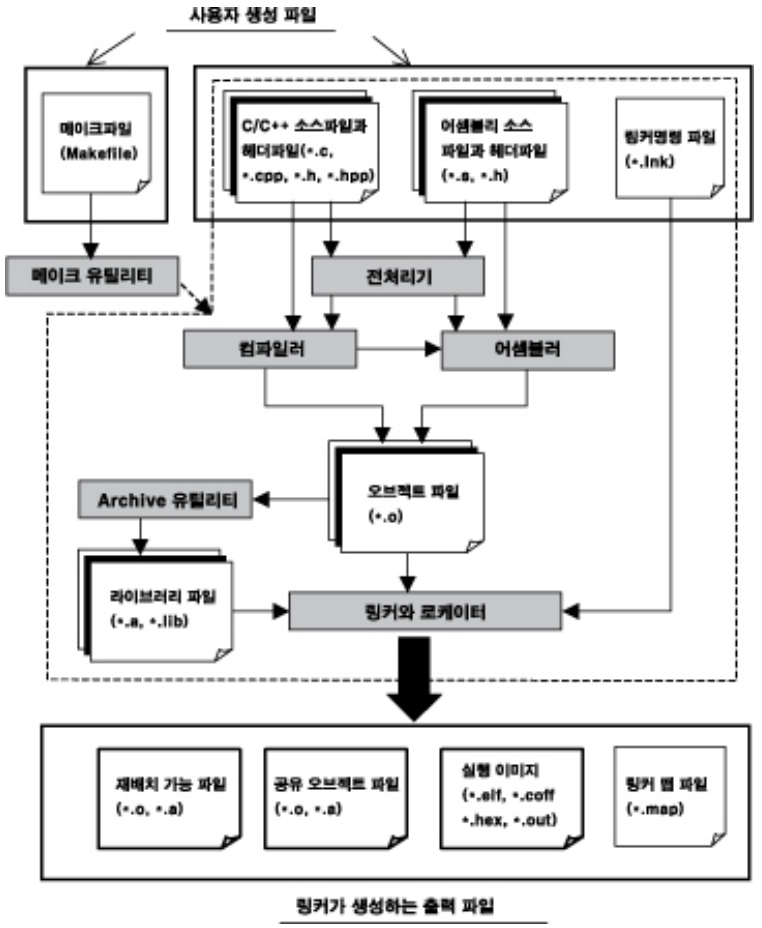
Host / Target System

- Host System (or Host PC)
  - 교차 컴파일러, 링커, 소스레벨 디버거
- Target System (or Target)
  - 링크로더, 모니터, 디버그 에이전트

- 교차 개발환경
  - 타겟에 실행이 가능하도록 개발
- 네이티브 개발환경 : 신경 쓸 필요 X

링커와 링크 과정

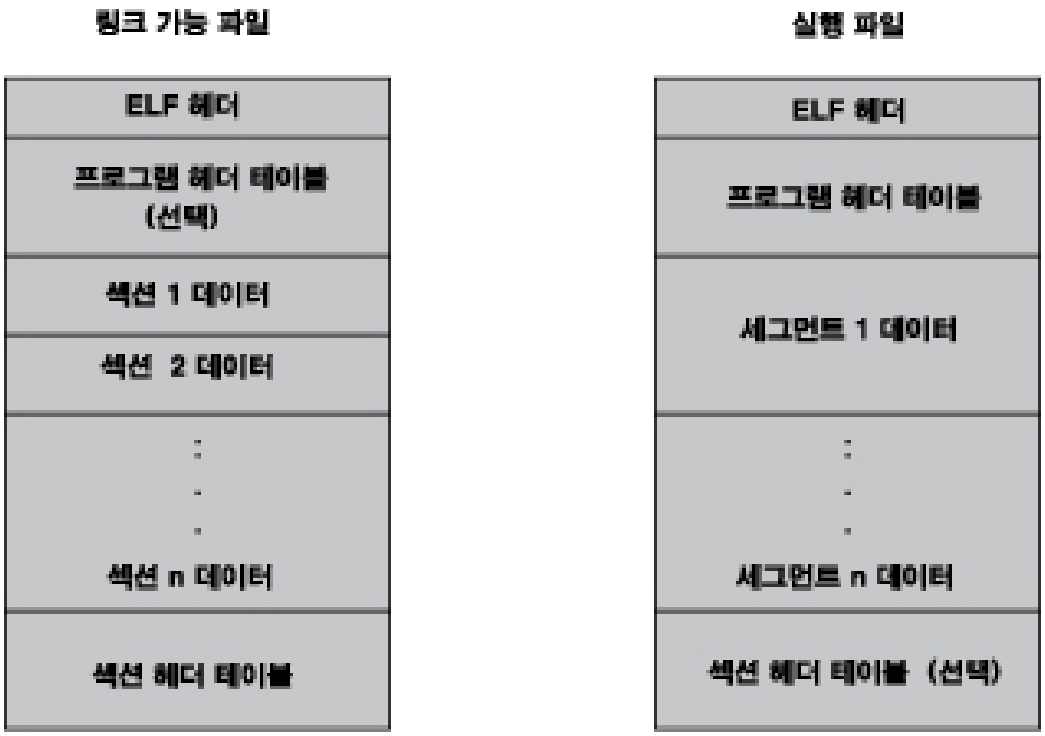
- 프로그램 작성 : C/C++, 소스파일, 헤더파일, makefile 작성
- 링커 : 오브젝트 파일을 입력으로 받아서 실행가능한 파일을 생성함
- 링커의 주 기능 : 여러 개의 오브젝트 파일을 입력으로 받아서
  - 재배치 가능한 *.o
  - 공유 *.o
  - 최종 실행 이미지



Object File (*.o)

- 파일 크기, 바이너리 코드와 데이터 크기, 오브젝트 파일을 생성하기 위해 사용한 소스파일명 등 일반적인 정보
- 프로세서 의존적인 바이너리 형태의 명령코드와 데이터
- 심볼 테이블과 심볼 재배치 테이블
- 디버거에서 필요로 하는 디버깅 정보

ELF


링크 가능 파일
ELF 헤더
프로그램 헤더 테이블
(선택)
섹션 1 데이터
섹션 2 데이터
:
.
.
섹션 n 데이터
섹션 헤더 테이블
실행 파일
ELF 헤더
프로그램 헤더 테이블
세그먼트 1 데이터
:
.
.
세그먼트 n 데이터
섹션 헤더 테이블 (선택)

ELF Header

```
soicem@ubuntu:~$ readelf -h /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x804bdee
  Start of program headers:          52 (bytes into file)
  Start of section headers:          119876 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         28
  Section header string table index: 27
```

출처 : https://blog.naver.com/knq1130/220688905842

Section

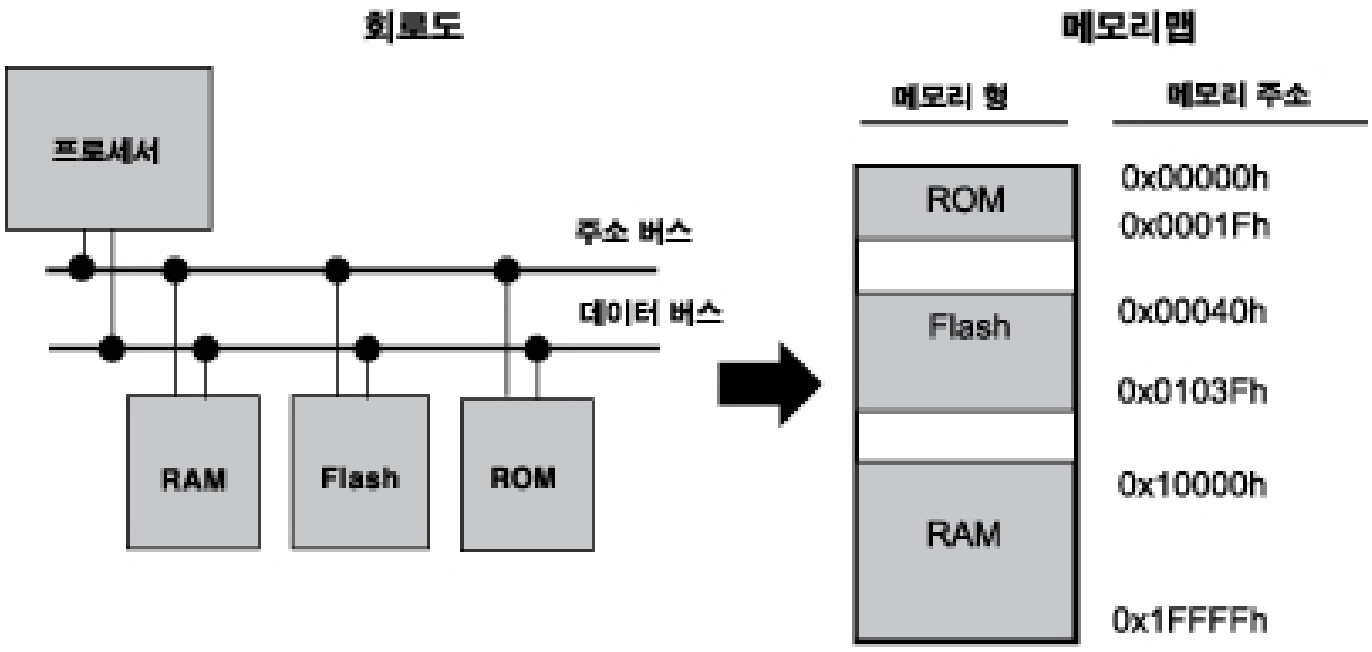
```
soicem@ubuntu:~$ readelf -S /bin/ls
There are 28 section headers, starting at offset 0x1d444:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ac 0001ac 00006c 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          08048218 000218 000820 10   A  6   1  4
  [ 6] .dynstr           STRTAB          08048a38 000a38 0005d5 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          0804900e 00100e 000104 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         08049114 001114 0000c0 00   A  6   2  4
  [ 9] .rel.dyn          REL             080491d4 0011d4 000038 08   A  5   0  4
  [10] .rel.plt          REL             0804920c 00120c 000380 08  AI  5  12  4
  [11] .init             PROGBITS        0804958c 00158c 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        080495b0 0015b0 000710 04  AX  0   0 16
  [13] .text             PROGBITS        08049cc0 001cc0 010104 00  AX  0   0 16
  [14] .fini             PROGBITS        08059dc4 011dc4 000014 00  AX  0   0  4
  [15] .rodata           PROGBITS        08059de0 011de0 0055d8 00   A  0   0 32
  [16] .eh_frame_hdr     PROGBITS        0805f3b8 0173b8 00074c 00   A  0   0  4
  [17] .eh_frame         PROGBITS        0805fb04 017b04 0048bc 00   A  0   0  4
  [18] .init_array       INIT_ARRAY      08065ef8 01cef8 000004 00  WA  0   0  4
  [19] .fini_array       FINI_ARRAY      08065efc 01cefc 000004 00  WA  0   0  4
  [20] .jcr              PROGBITS        08065f00 01cf00 000004 00  WA  0   0  4
  [21] .dynamic          DYNAMIC         08065f04 01cf04 0000f8 08  WA  6   0  4
  [22] .got              PROGBITS        08065ffc 01cffc 000004 04  WA  0   0  4
  [23] .got.plt          PROGBITS        08066000 01d000 0001cc 04  WA  0   0  4
  [24] .data             PROGBITS        080661e0 01d1e0 000160 00  WA  0   0 32
  [25] .bss              NOBITS          08066340 01d340 000c34 00  WA  0   0 64
  [26] .gnu_debuglink    PROGBITS        00000000 01d340 000008 00      0   0  1
  [27] .shstrtab         STRTAB          00000000 01d348 0000fc 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

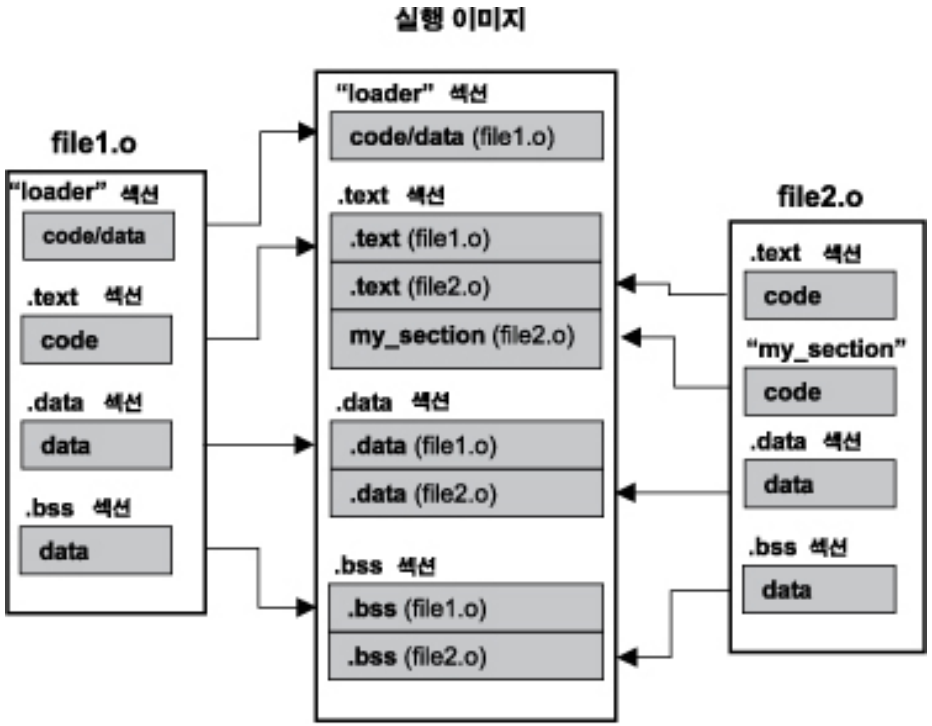
출처 : https://blog.naver.com/knq1130/220688905842

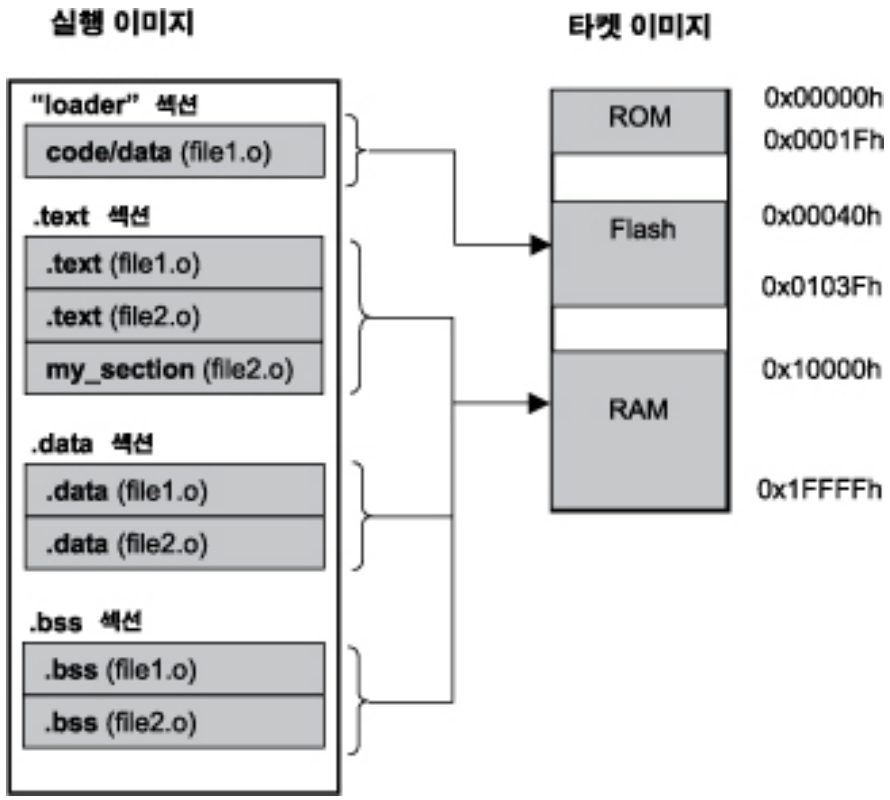
Program Header

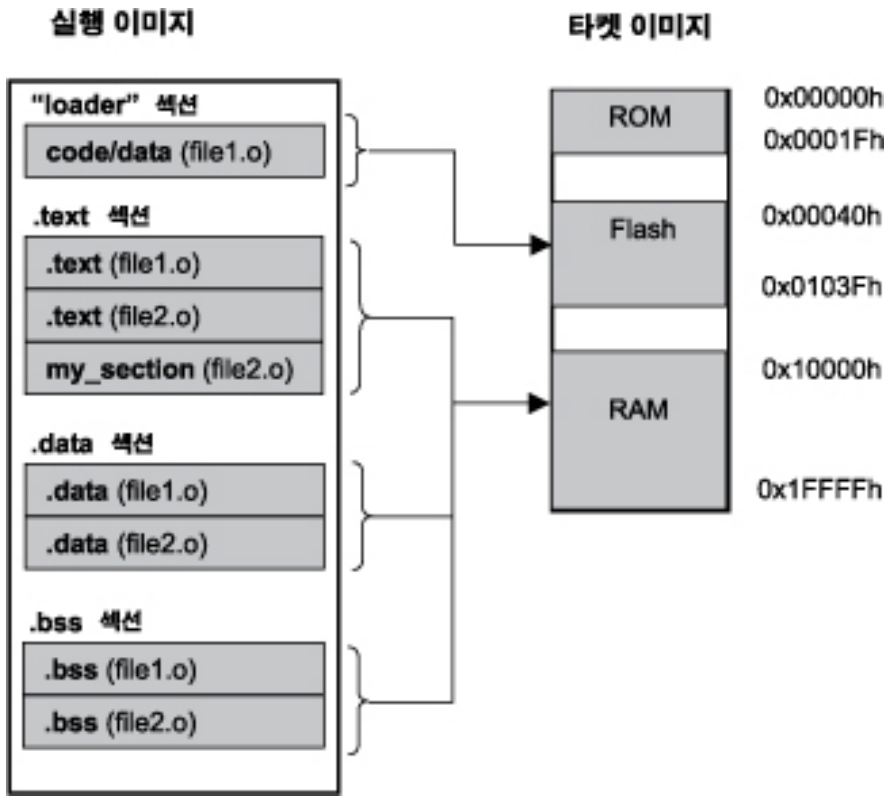
```
soicem@ubuntu:~/Desktop/packer$ readelf -l /bin/ls

Elf file type is EXEC (Executable file)
Entry point 0x804bdee
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x1c3c0 0x1c3c0 R E 0x1000
  LOAD           0x01cef8 0x08065ef8 0x08065ef8 0x00448 0x0107c RW  0x1000
  DYNAMIC        0x01cf04 0x08065f04 0x08065f04 0x000f8 0x000f8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0173b8 0x0805f3b8 0x0805f3b8 0x0074c 0x0074c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x01cef8 0x08065ef8 0x08065ef8 0x00108 0x00108 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version .gnu.versio
n_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
```

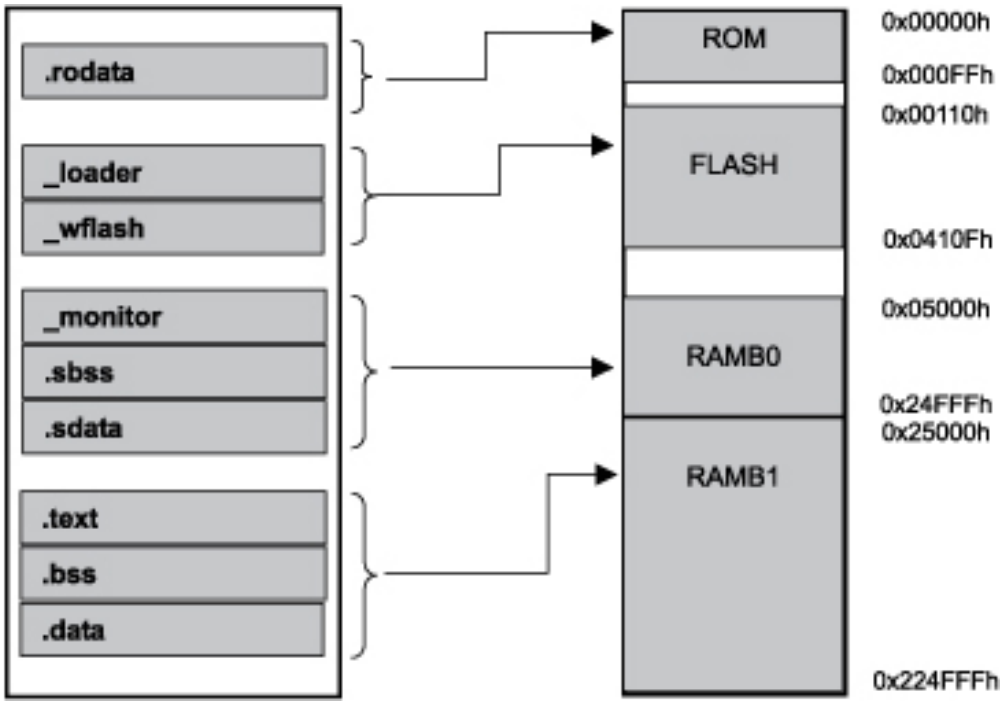
출처 : https://blog.naver.com/knq1130/220688905842


회로도
프로세서
주소 버스
데이터 버스
RAM
Flash
ROM
메모리맵
메모리 형
메모리 주소
ROM
Flash
RAM
0x00000h
0x0001Fh
0x00040h
0x0103Fh
0x10000h
0x1FFFFh


실행 이미지
file1.o
"loader" 섹션
code/data
.text 섹션
code
.data 섹션
data
.bss 섹션
data
"loader" 섹션
code/data (file1.o)
.text 섹션
.text (file1.o)
.text (file2.o)
my_section (file2.o)
.data 섹션
.data (file1.o)
.data (file2.o)
.bss 섹션
.bss (file1.o)
.bss (file2.o)
file2.o
.text 섹션
code
"my_section"
code
.data 섹션
data
.bss 섹션
data


실행 이미지
타켓 이미지
"loader" 섹션
code/data (file1.o)
.text 섹션
.text (file1.o)
.text (file2.o)
my_section (file2.o)
.data 섹션
.data (file1.o)
.data (file2.o)
.bss 섹션
.bss (file1.o)
.bss (file2.o)
ROM
Flash
RAM
0x00000h
0x0001Fh
0x00040h
0x0103Fh
0x10000h
0x1FFFFh


실행 이미지
타켓 이미지
"loader" 섹션
code/data (file1.o)
.text 섹션
.text (file1.o)
.text (file2.o)
my_section (file2.o)
.data 섹션
.data (file1.o)
.data (file2.o)
.bss 섹션
.bss (file1.o)
.bss (file2.o)
ROM
Flash
RAM
0x00000h
0x0001Fh
0x00040h
0x0103Fh
0x10000h
0x1FFFFh


.rodata
_loader
_wflash
_monitor
.sbss
.sdata
.text
.bss
.data
ROM
FLASH
RAMB0
RAMB1
0x00000h
0x000FFh
0x00110h
0x0410Fh
0x05000h
0x24FFFh
0x25000h
0x224FFFh

Lecture 3

2018년 1학기

임베디드 시스템 초기화

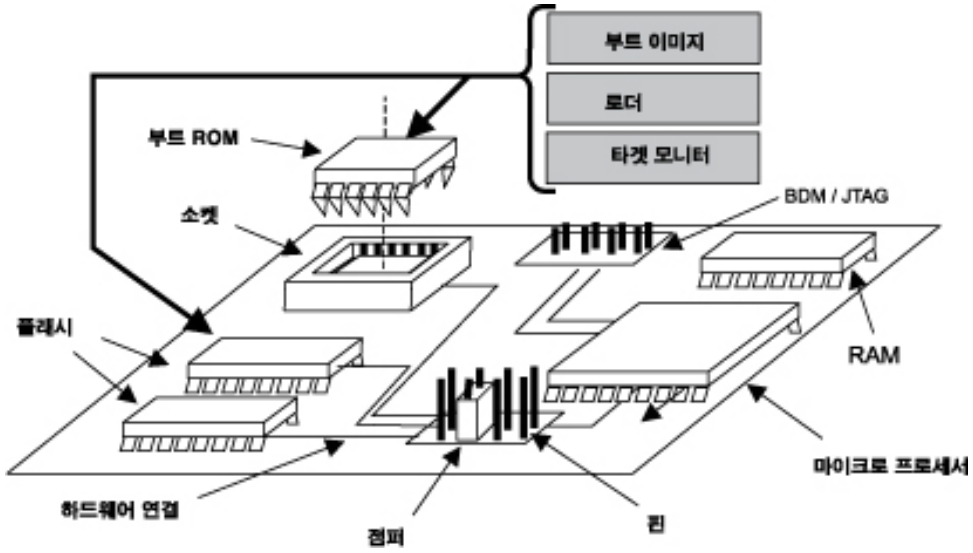
- C언어 → Hello World!
- 임베디드 개발환경 → Hello World!
  - 타겟 시스템에 실행 이미지 또는 프로그램을 어떻게 로드 할 것인가
  - 메모리의 어느 주소에 이미지를 로드 할 것인가
  - 어떻게 실행할 것인가
  - 정상동작 여부 확인을 어떻게 할 것인가

타겟 임베디드 시스템

- 타겟 시스템에서 실행할 프로그램들은 호스트로부터 받아서 실행 (로딩 또는 이미지 로딩)
- 최종 임베디드 S/W는 ROM 또는 플래쉬 메모리에 저장

타겟 임베디드 시스템

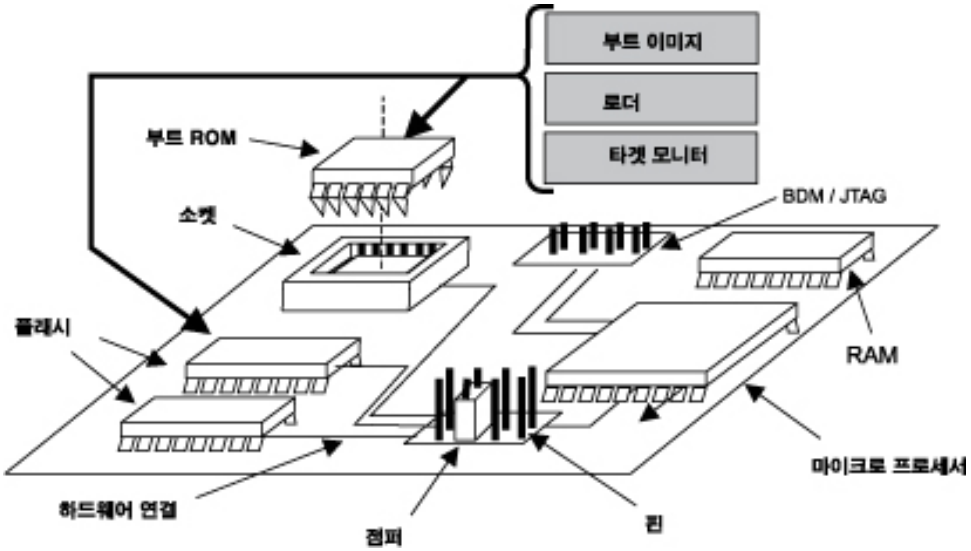
- 임베디드 로더
  - 로더를 동작시켜 호스트로부터 이미지 다운로드
  - 호스트 유틸리티와 통신
  - ROM에 프로그램 하여 저장
  - 부트 이미지 : 타겟 하드웨어를 안정적인 상태로 초기화하는 일, 준비작업
  - 부트가 초기화 작업 마치면 로더가 실행

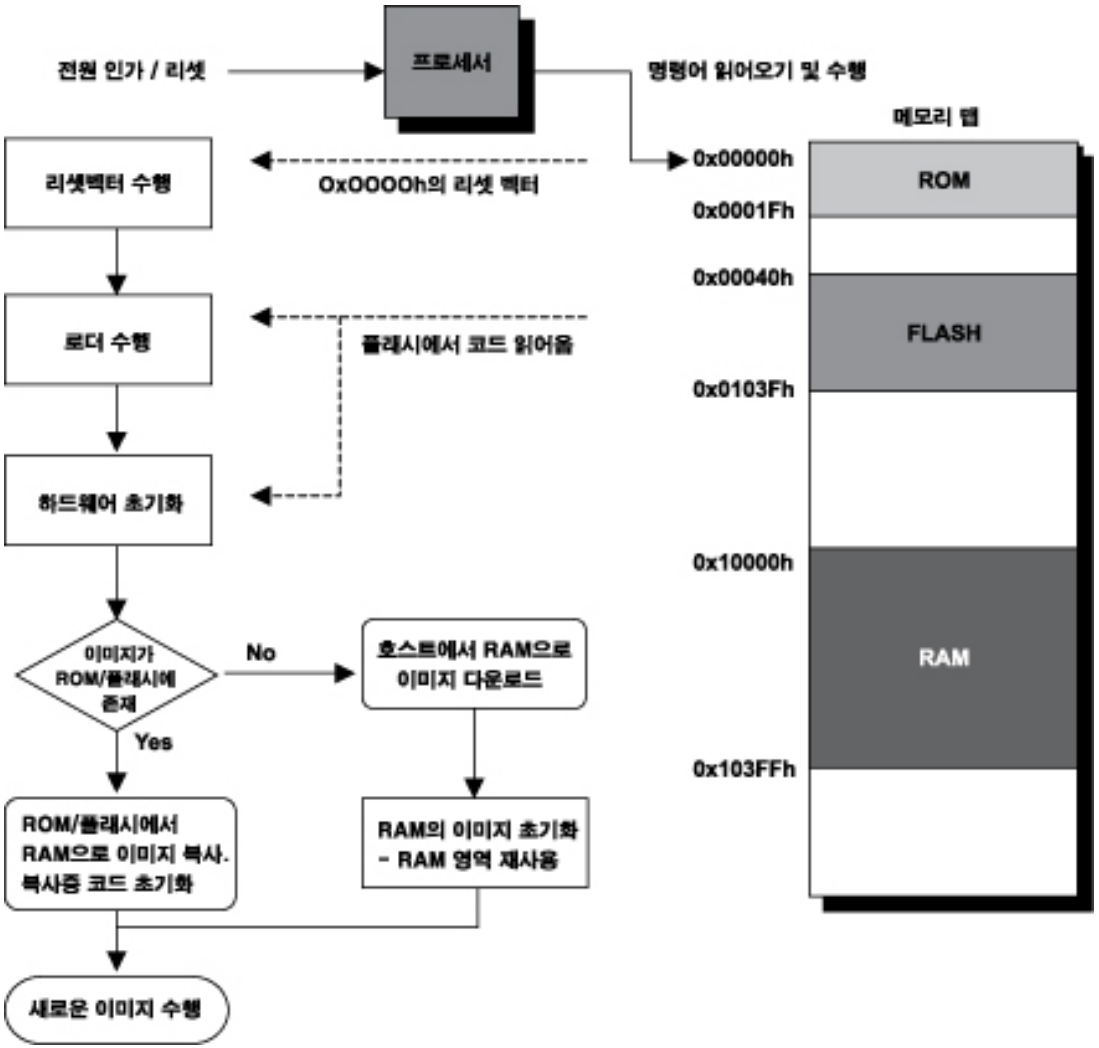


타겟 임베디드 시스템

- 임베디드 로더
  - 호스트와 타겟 사이의 데이터 전송 프로토콜, 통신 파라미터 등이 결정되어야 함

  예 : 로더가 먼저 호스트 유틸리티에 이미지 전송 요청을 하면 호스트 유틸리티에서 이미지의 크기와 이미지를 전송해주고 전송이 끝나면 로더는 확인 패킷을 보내준다.
  - 통신 prm : 데이터 전송율, 패킷 당 데이터 크기 등




전원 인가 / 리셋
프로세서
명령어 읽어오기 및 수행
메모리 맵
0x00000h
ROM
0x0001Fh
0x00040h
FLASH
0x0103Fh
0x10000h
RAM
0x103FFh
리셋벡터 수행
OxOOOOh의 리셋 벡터
로더 수행
플래시에서 코드 읽어옴
하드웨어 초기화
이미지가 ROM/플래시에 존재
No
Yes
호스트에서 RAM으로 이미지 다운로드
ROM/플래시에서 RAM으로 이미지 복사. 복사중 코드 초기화
RAM의 이미지 초기화 - RAM 영역 재사용
새로운 이미지 수행