EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv 1 과정

제 5기

2023. 06. 10

Lee sungkil

C+ inline assembler (asm)

1) Inline assembler

Inline은 함수 호출을 통하여 코드 흐름을 제어하지 않고, 함수 내부에 코드를 삽입하므로 함수 호출로 인한 부하를 줄일 수 있으며, " inline function"이라고도 한다.

inline assembly라 함은 어셈블리 명령들을 inline 함수로 작성하는 것이다.
인라인 어셈블리는 실행속도 향상을 위해서 사용하며, 시스템 프로그래밍에서 자주 사용된다.

C언어에서는 asm 키워드를 사용하여 어셈블리 명령어를 코딩한다.
C언어와 어셈블리 명령어를 혼합할 수 있으므로, 빠른 실행처리는 어셈블리로 수행하고, 결과 데이터는 C언어의 변수들로 확인 가능한 장점이 있다.

2) Inline assembly 문법

asm _volatile_ (asms : output : input : clobber)

asm : 다음에 나오는 것이 inline assembly 임을 나타낸다. ANSI엔 _asm_으로만 정의되어 있으므로 asm과 같은 키워드는 사용하지 않는 것이 바람직하다.
volatile : 이 키워드를 사용하면 컴파일러는 프로그래머가 입력한 그래도 남겨두게 된다. 즉 최적화 나 위치를 옮기는 등의 일은 하지 않는다. 예를 들어 output 변수 중 하나가 inline assembly엔 명시되어 있지만 다른 곳에서 사용되지 않는다고 판단되면 컴파일러는 이 변수를 알아서 잘 없애주기도 한다. 이런 경우 이런 것을 고려해 프로그램을 짰다면 상관없겠지만 만에 하나 컴파일러가 자동으로 해준 일 때문에 버그가 발생할 수도 있다. 그러므로 _volatile_ 키워드를 사용해 주는 것이 좋다.

Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
->  0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add    x29, sp, #0x10
    0x100003f70 <+12>: mov    w8, #0x0
    0x100003f74 <+16>: str    w8, [sp, #0x8]
    0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

fp	0	0	0	210f8000	16fdff590
sp	0	0	0	0	16fdff330

```c
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```

Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
    0x100003f64 <+0>:  sub   sp, sp, #0x20
->  0x100003f68 <+4>:  stp   x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add   x29, sp, #0x10
    0x100003f70 <+12>: mov   w8, #0x0
    0x100003f74 <+16>: str   w8, [sp, #0x8]
    0x100003f78 <+20>: stur  wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl    0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl    0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr   w0, [sp, #0x8]
    0x100003f88 <+36>: ldp   x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add   sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

```c
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```

fp	0	0	0	210f8000	x29	16fdff590
	0	0	0	0		16fdff330
	6fdff590	1	9d63bf10	1518001		16fdff320
sp						16fdff310

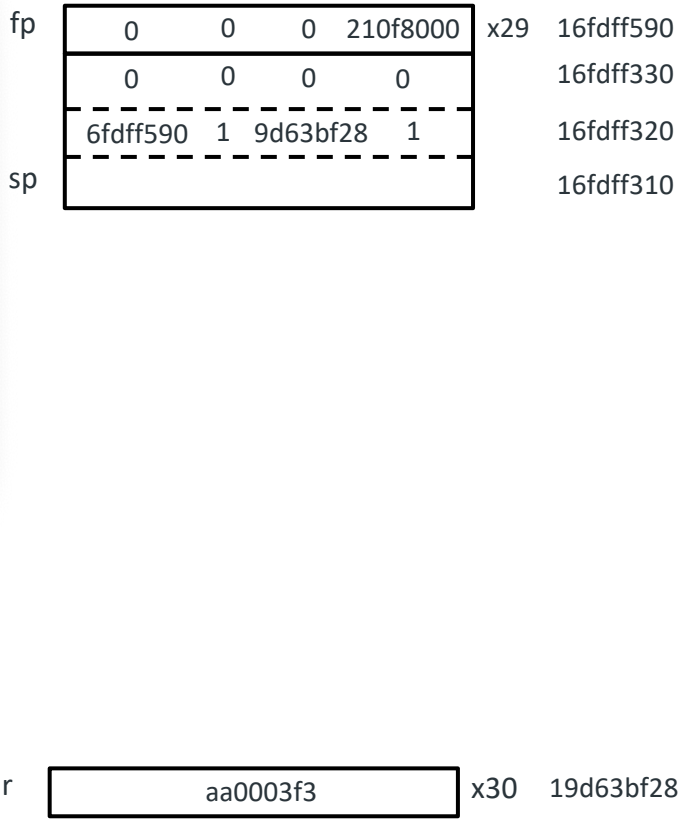
lr	aa0003f3	x30	19d63bf28

Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
    0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
->  0x100003f6c <+8>:  add    x29, sp, #0x10
    0x100003f70 <+12>: mov    w8, #0x0
    0x100003f74 <+16>: str    w8, [sp, #0x8]
    0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

```
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```

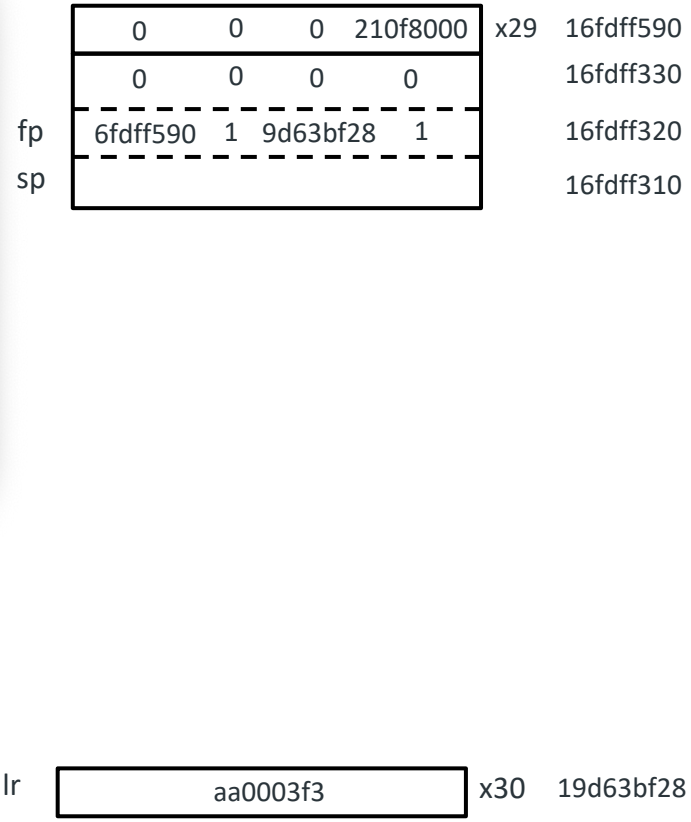


Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
    0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add    x29, sp, #0x10
->  0x100003f70 <+12>: mov    w8, #0x0
    0x100003f74 <+16>: str    w8, [sp, #0x8]
    0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

```
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```



Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
    0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add    x29, sp, #0x10
    0x100003f70 <+12>: mov    w8, #0x0
->  0x100003f74 <+16>: str    w8, [sp, #0x8]
    0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

```
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```

	0	0	0	210f8000	x29	16fdff590
	0	0	0	0		16fdff330
fp	6fdff590	1	9d63bf28	1		16fdff320
sp	1000	1	3f64	1		16fdff310

w8			0
lr	aa0003f3	x30	19d63bf28

Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
    0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add    x29, sp, #0x10
    0x100003f70 <+12>: mov    w8, #0x0
    0x100003f74 <+16>: str    w8, [sp, #0x8]
->  0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

	0	0	0	210f8000	x29	16fdff590
	0	0	0	0		16fdff330
fp	6fdff590	1	9d63bf28	1		16fdff320
sp	1000	1	0	1		16fdff310

```c
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```

w8			0
lr	aa0003f3	x30	19d63bf28

Inline_asm.c 코드 분석

```
lldb inline_asm
inline_asm`main:
    0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add    x29, sp, #0x10
    0x100003f70 <+12>: mov    w8, #0x0
    0x100003f74 <+16>: str    w8, [sp, #0x8]
    0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
->  0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```

	0	0	0	210f8000	x29	16fdff590
	0	0	0	0		16fdff330
fp	6fdff590	1	9d63bf28	1		16fdff320
sp	1000	1	0	0		16fdff310

```c
int main (void)
{
        can_I_execute_asm();
        yes_you_can_execute_asm();

        return 0;
}
```

w8			0
lr	aa0003f3	x30	19d63bf28

Inline_asm.c 코드 분석

```c
void can_I_execute_asm (void)
{
register unsigned int *x8 asm("x8") = 0;
x8 = arr;

asm volatile("mov w1, #0x3");

asm volatile("lsl w2, w1, #0x2");
asm volatile("mov w4, #0x2");

asm volatile("lsl w3, w2, w4");
asm volatile("add w3, w3, w1");

asm volatile("str w1, [x8]");
asm volatile("str x2, [x8, #0x4]");
asm volatile("str w3, [x8, #0x8]");

print_arr_info(arr);
}
```

lldb inline_asm

```
inline_asm`can_I_execute_asm:
->  0x100003ebc <+0>:  sub    sp, sp, #0x20
    0x100003ec0 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003ec4 <+8>:  add    x29, sp, #0x10
    0x100003ec8 <+12>: str    xzr, [sp, #0x8]
    0x100003ecc <+16>: adrp   x0, 5
    0x100003ed0 <+20>: add    x0, x0, #0x0              ; arr
    0x100003ed4 <+24>: mov    x8, x0
    0x100003ed8 <+28>: str    x8, [sp, #0x8]
    0x100003edc <+32>: mov    w1, #0x3
    0x100003ee0 <+36>: lsl    w2, w1, #2
    0x100003ee4 <+40>: mov    w4, #0x2
    0x100003ee8 <+44>: lsl    w3, w2, w4
    0x100003eec <+48>: add    w3, w3, w1
    0x100003ef0 <+52>: str    w1, [x8]
    0x100003ef4 <+56>: stur   x2, [x8, #0x4]
    0x100003ef8 <+60>: str    w3, [x8, #0x8]
    0x100003efc <+64>: bl     0x100003e40               ; print_arr_info
    0x100003f00 <+68>: ldp    x29, x30, [sp, #0x10]
    0x100003f04 <+72>: add    sp, sp, #0x20
    0x100003f08 <+76>: ret
```

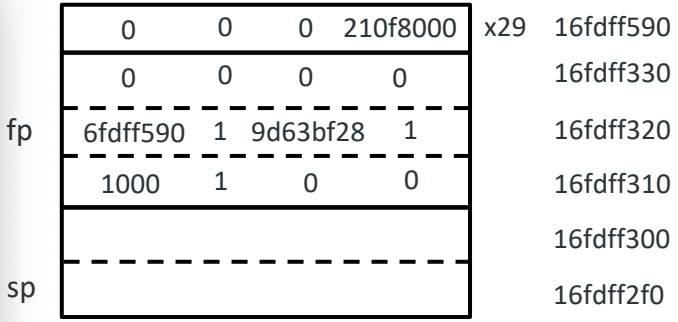
Inline_asm.c 코드 분석

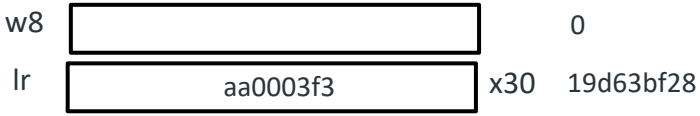
EDDI
Electronic Design Development Institute

lldb inline_asm

```
inline_asm`can_I_execute_asm:
    0x100003ebc <+0>:  sub    sp, sp, #0x20
->  0x100003ec0 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003ec4 <+8>:  add    x29, sp, #0x10
    0x100003ec8 <+12>: str    xzr, [sp, #0x8]
    0x100003ecc <+16>: adrp   x0, 5
    0x100003ed0 <+20>: add    x0, x0, #0x0              ; arr
    0x100003ed4 <+24>: mov    x8, x0
    0x100003ed8 <+28>: str    x8, [sp, #0x8]
    0x100003edc <+32>: mov    w1, #0x3
    0x100003ee0 <+36>: lsl    w2, w1, #2
    0x100003ee4 <+40>: mov    w4, #0x2
    0x100003ee8 <+44>: lsl    w3, w2, w4
    0x100003eec <+48>: add    w3, w3, w1
    0x100003ef0 <+52>: str    w1, [x8]
    0x100003ef4 <+56>: stur   x2, [x8, #0x4]
    0x100003ef8 <+60>: str    w3, [x8, #0x8]
    0x100003efc <+64>: bl     0x100003e40               ; print_arr_info at inline_asm.c:9
    0x100003f00 <+68>: ldp    x29, x30, [sp, #0x10]
    0x100003f04 <+72>: add    sp, sp, #0x20
    0x100003f08 <+76>: ret
(lldb)
```

```
void can_I_execute_asm (void)
{
register unsigned int *x8 asm("x8") = 0;
x8 = arr;
```





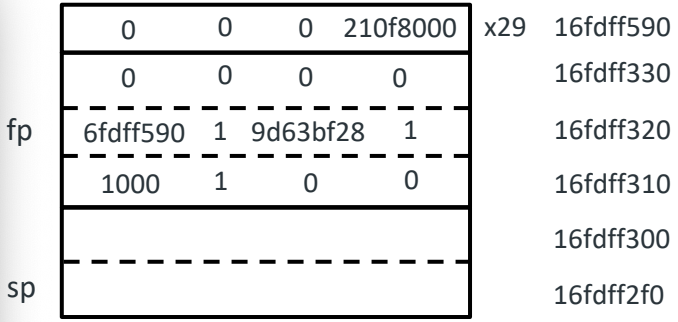
Inline_asm.c 코드 분석

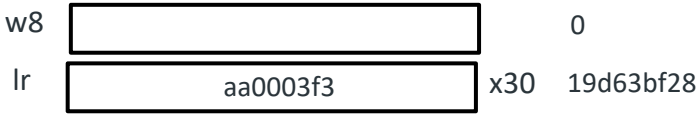
EDDI
Electronic Design Development Institute

lldb inline_asm

```
inline_asm`can_I_execute_asm:
    0x100003ebc <+0>:  sub    sp, sp, #0x20
->  0x100003ec0 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003ec4 <+8>:  add    x29, sp, #0x10
    0x100003ec8 <+12>: str    xzr, [sp, #0x8]
    0x100003ecc <+16>: adrp   x0, 5
    0x100003ed0 <+20>: add    x0, x0, #0x0              ; arr
    0x100003ed4 <+24>: mov    x8, x0
    0x100003ed8 <+28>: str    x8, [sp, #0x8]
    0x100003edc <+32>: mov    w1, #0x3
    0x100003ee0 <+36>: lsl    w2, w1, #2
    0x100003ee4 <+40>: mov    w4, #0x2
    0x100003ee8 <+44>: lsl    w3, w2, w4
    0x100003eec <+48>: add    w3, w3, w1
    0x100003ef0 <+52>: str    w1, [x8]
    0x100003ef4 <+56>: stur   x2, [x8, #0x4]
    0x100003ef8 <+60>: str    w3, [x8, #0x8]
    0x100003efc <+64>: bl     0x100003e40               ; print_arr_info at inline_asm.c:9
    0x100003f00 <+68>: ldp    x29, x30, [sp, #0x10]
    0x100003f04 <+72>: add    sp, sp, #0x20
    0x100003f08 <+76>: ret
(lldb)
```

```
void can_I_execute_asm (void)
{
register unsigned int *x8 asm("x8") = 0;
x8 = arr;
```





Inline_asm.c 코드 분석

```c
#include <stdio.h>

#define MAX 6

unsigned int arr[MAX] = { 1, 2, 3, 4, 5 };
unsigned int other_arr[MAX];
```

Inline_asm.c 코드 분석

```c
void print_arr_info(unsigned int *arr)
{
        int i;

        for (i = 0; i < MAX; i++)
        {
                printf("arr[%d] = %d\n", i, arr[i]);
        }
}
```

lldb inline_asm

```
inline_asm`main:
->  0x100003f64 <+0>:  sub    sp, sp, #0x20
    0x100003f68 <+4>:  stp    x29, x30, [sp, #0x10]
    0x100003f6c <+8>:  add    x29, sp, #0x10
    0x100003f70 <+12>: mov    w8, #0x0
    0x100003f74 <+16>: str    w8, [sp, #0x8]
    0x100003f78 <+20>: stur   wzr, [x29, #-0x4]
    0x100003f7c <+24>: bl     0x100003ebc              ; can_I_execute_asm
    0x100003f80 <+28>: bl     0x100003f0c              ; yes_you_can_execute_asm
    0x100003f84 <+32>: ldr    w0, [sp, #0x8]
    0x100003f88 <+36>: ldp    x29, x30, [sp, #0x10]
    0x100003f8c <+40>: add    sp, sp, #0x20
    0x100003f90 <+44>: ret
(lldb)
```