기반 작업
(3) struct gcc_target 이란?

정원교

2004년 2월 23일

목 차

제 1 절 9 주째 강의를 시작하며

지난 8 주 강의는 저로써는 약간 실수를 한 강의라고 생각을 합니다. 무엇보다 주제를 접근하는데 있어서 조금의 착오가 있었으며 이에 대한 전달 방법 또한 미숙했습니다. 구체적인 GCC 내부 수행들은 제가 세세히 설명할 필요까지는 없을 것 같습니다. 대신 GCC 내에서 사용되는 구조체들과 함수들의 쓰임새 등을 좀 더 자세히 기술하면 그것이 가장 좋은 상태라고 지금 생각합니다. 이번 주는 GCC 내에서 포함하고 있는 또 다른 구조체인 struct gcc_target 에 대해서 알아 보도록 하겠습니다.

제 2 절 struct gcc_target 이란?

struct gcc_target 구조체는 GCC target 에 대해 기술하는 정보를 포함하고 있습니다. 현재로써는 불완전한 형태이지만 조만간 대부분 혹은 모든 target machine 과 target O/S specific 정보까지 포함하는 형태로 발전할 것입니다.

이 구조체는 자신의 초기화자를 가지고 있으며 그것은 target-def.h 에 정의되어 있습니다. 그 매크로의 이름은 TARGET_INITIALIZER 로써 내부에 많은 작은 매크로들을 포함하고 있는 큰 매크로입니다. 이 매크로에 대해서는 아래에서 다시 언급할 것입니다.

각각의 작은 매크로들은 하나의 구조체 component 를 초기화하고 각각의 기본값을 가지고 있씁니다. 각 target 은 target.h 와 target-def.h 를 포함하는 파일을 가지고 있어야 하며 부적절한 매크로를 undefine 하고 적절한 대체품으로 재정의하여 기본값들을 재조정해주어야 합니다. 그리고 그것을 정의하는 파일은 아래와 같은 "targetm" 정의를 포함하고 있어야 합니다.

struct gcc_target targetm = TARGET_INITIALIZER;

이러한 방법을 사용함으로써 GCC target 를 정의하는데 있어서 모든 것을 함께 처리할 수 있도록 합니다. 대부분의 target 들에게 적당한 기본값들을 제공함으로써, 우리는 수십개의 target 설정 파일들을 편집할 필요없이 새로운 아이템들을 쉽게 추가할 수 있습니다. 또한 GCC 소스에서 뿔뿔이 흩어지게 될 수 있는 조건 컴파일(conditional compilation)의 량을 점차적으로 줄이는 역활도 할 수 있습니다.

이제 struct gcc_target 를 구성하는 요소들에 대해서 알아보도록 하겠습니다.

제 3 절 struct gcc_target 의 구성요소

그럼 이 구조체를 구성하는 요소에 대해서 알아 보도록 하겠습니다. 아래의 구조체가 원형을 가르킵니다.

```
struct gcc_target
{
  struct asm_out
  {
    const char *open_paren, *close_paren;

    const char *byte_op;
    struct asm_int_op
    {
      const char *hi;
      const char *si;
      const char *di;
      const char *ti;
    } aligned_op, unaligned_op;

    bool (* integer) PARAMS ((rtx x, unsigned int size, int aligned_p));
    void (* function_prologue) PARAMS ((FILE *, HOST_WIDE_INT));
    void (* function_end_prologue) PARAMS ((FILE *));
    void (* function_begin_epilogue) PARAMS ((FILE *));
    void (* function_epilogue) PARAMS ((FILE *, HOST_WIDE_INT));
    void (* named_section) PARAMS ((const char *, unsigned int));
    void (* exception_section) PARAMS ((void));
    void (* eh_frame_section) PARAMS ((void));
    void (* constructor) PARAMS ((rtx, int));
    void (* destructor) PARAMS ((rtx, int));
  } asm_out;

  struct sched
  {
    int (* adjust_cost) PARAMS ((rtx insn, rtx link, rtx def_insn,
```

```
                                int cost));
    int (* adjust_priority) PARAMS ((rtx, int));
    int (* issue_rate) PARAMS ((void));
    int (* variable_issue) PARAMS ((FILE *, int, rtx, int));
    void (* md_init) PARAMS ((FILE *, int, int));
    void (* md_finish) PARAMS ((FILE *, int));
    int (* reorder)  PARAMS ((FILE *, int, rtx *, int *, int));
    int (* reorder2) PARAMS ((FILE *, int, rtx *, int *, int));
    rtx (* cycle_display) PARAMS ((int clock, rtx last));
  } sched;

  tree (* merge_decl_attributes) PARAMS ((tree, tree));
  tree (* merge_type_attributes) PARAMS ((tree, tree));

  const struct attribute_spec *attribute_table;

  int (* comp_type_attributes) PARAMS ((tree type1, tree type2));
  void (* set_default_type_attributes) PARAMS ((tree type));
  void (* insert_attributes) PARAMS ((tree decl, tree *attributes));
  bool (* function_attribute_inlinable_p) PARAMS ((tree fndecl));
  bool (* ms_bitfield_layout_p) PARAMS ((tree record_type));
  void (* init_builtins) PARAMS ((void));
  rtx (* expand_builtin) PARAMS ((tree exp, rtx target, rtx subtarget,
                                  enum machine_mode mode, int ignore));
  unsigned int (* section_type_flags) PARAMS ((tree, const char *, int));

  bool have_named_sections;
  bool have_ctors_dtors;

  bool (* cannot_modify_jumps_p) PARAMS ((void));
};
```

각 구성요소에 대한 설명을 하겠습니다.

- `asm_out`
  target 을 위한 assembler 를 output 하는 함수들.
  - `open_paren, close_paren`
    asm 표현식 그룹화를 위한 여닫이 괄호.
  - `byte_op`
    여러 종류의 정수 object 를 생성하기 위한 어셈블러 명령어들.
  - `* integer`
    값이 X 로 주어진 정수 object 를 위한 어셈블러 code 를 output 함. SIZE 는 바이트 크기의 object 크기이며 ALIGNED_P 는 align 되었는지에 대해 가르킵니다. 만약 성공적으로 수행되었다면 true 를 반환합니다. BYTE_OP 와 ALIGNED_OP, UNALIGNED_OP 가 NULL 인 경우만 다루게 됩니다.
  - `* function_prologue`
    함수의 시작부분 관련 어셈블러 code 를 output.
  - `* function_end_prologue`
    함수의 끝부분 관련 어셈블러 code 를 output.
  - `* function_begin_epilogue`
    epilogue 의 시작부분 관련 어셈블러 code 를 output.
  - `* function_epilogue`
    함수의 출구부분 관련 어셈블러 code 를 output.

  - `* named_section`
    arbitrary section NAME 을 FLAGS 에서 지정한 attribute 로 변환.
  - `* exception_section`
    Exception table 을 가지고 있는 (hold) section 을 바꿉니다.
  - `* eh_frame_section`
    Exception frame 들을 가지고 있는 section 을 바꿉니다.
  - `* constructor`
    주어진 priority 를 가진 symbol 을 위한 constructor 를 output 합니다.
  - `* destructor`
    주어진 priority 를 가진 symbol 을 위한 desctructor 를 output 합니다.

- `sched`
  명령어 스케줄링 관련 함수들.
  - `* adjust_cost`
    주어진 현재 명령어 INSN 의 COST 를 계산하고 의존성 LINK 를 통한 DEP_INSN 와의 관계에 기반하여 새로운 cost 를 반환합니다. 기본값은 조정 (adjustment) 이 없는 것으로 합니다.
  - `* adjust_priority`
    당신이 맞추고자 하는 명령어의 priority 로 조정합니다. 새로운 priority 를 반환합니다.
  - `* issue_rate`
    같은 machine cycle 내에서 스케줄될 수 있는 명령어의 최대수를 반환하는 함수. 이것은 전체 컴파일 과정에서 상수 (불편수) 여야 합니다. 기본값은 1 입니다.
  - `* variable_issue`
    이 명령어가 얼마나 영향을 미치는지 혹은 우리가 이 cycle 에서 emit 할 수 있는 명령어가 더 얼마나 있는지를 계산합니다. 기본값은 모든 cost 가 같은 것으로 표현됩니다.
  - `* md_init`
    machine-dependent scheduling code 를 초기화합니다.
  - `* md_finish`
    machine-dependent scheduling code 를 마무리합니다.
  - `* reorder, * reorder2`
    두개의 다른 곳으로 명령어들을 machine-dependent fashin 으로 재정리합니다. 기본값은 이 수행을 하지 않는 것입니다.
  - `* cycle_display`
    cycle_display 는 현재 cycle 에 대해 assembly stream 으로 data 를 방출할 수 있도록 하는 함수를 가르키는 포인터이다. 인자들로는 CLOCK (방출할 data), 그리고 LAST (우리가 building 하고 있는 새로운 chain 에서의 마지막 insn) 들이 있다. 새로운 LAST 를 반환합니다. 기본값으로는 수행하지 않는 것으로 되어 있습니다.

- `* merge_decl_attributes`
  주어진 두개의 decl 들의 attribute 들을 merge 하여 결과를 반환합니다.

- `* merge_type_attributes`
  주어진 두개의 type 들의 attribute 들을 merge 하여 결과를 반환합니다.

- `*attribute_table`
  그것을 다룰 machine attribute 들과 function 들에 대한 테이블.

- `* comp_type_attributes`
  만약 TYPE1 과 TYPE2 가 호환성이 없다면 0 을 반환하고 만약 호환성이 있다면 1 을 반환하며 거의 호환성이 있다고 판단된다면 2 를 반환합니다. (거의 호환성이 있다는 말은 생성될 때 경고 메세지를 발생시킬 수 있다는 것을 말합니다.)

- * set_default_type_attributes
  새로이 정의된 TYPE 을 위한 기본 attribute 들을 할당합니다.
- * insert_attributes
  새로이 생성된 DECL 상에 attribute 들을 삽입합니다.
- * function_attribute_inlinable_p
  만약 FNDECL (적어도 하나의 machine attribute 를 가지고 있는) 가 그것의 machine attribute 에도 불구하고 inline 되어 질 수 있다면 true 를 반환하고 그렇지 않을 경우 false 를 반환합니다.
- * ms_bitfield_layout_p
  만약 RECORD_TYPE 내부의 bitfield 들이 Microsoft Visual C++ bitfield layout rule 들을 뒤에 따라 가지고 있어야 한다면 true 를 반환합니다.
- * init_builtins
  Target-specific built-in 함수들을 설정합니다.
- * expand_builtin
  Target-specific builtin 을 확장합니다.
- * section_type_flags
  주어진 decl 와 section name, 그리고 decl 초기화자가 reloc 들을 가지고 있는지가 section 을 위한 attribute 들을 선택합니다.
- have_named_sections
  만약 arbitrary section 이 지원되면 true.
- have_ctors_dtors
  만약 "native" constructor 들과 destructor 들이 지원된다면 true, 만약 우리가 작업을 위해 collect2 를 사용하고 있다면 false.
- * cannot_modify_jumps_p
  현재 컴파일 시점에서 존재하는 jump 혹은 존재하지 않는 jump 를 대체할 새로운 jump 가 생성되어 질수 없다면 true.

제 4 절 struct gcc_target 의 기본 초기화자

struct gcc_target 를 정의할 때는 꼭 TARGET_INITIALIZER 매크로를 사용하여 정의한다고 앞에서 언급하였습니다. 실제로 i386 을 위한 gcc_target 구조체는 $prefix/gcc/config/i386/i386.c 에 선언되어 있습니다. 아래와 같이 선언되어 있는데, 이것으로 보면 TARGET_INITIALIZER 매크로가 struct gcc_target 에 대한 모든 정보를 포함하고 있는 매크로라는 사실을 알 수 있습니다.

```
struct gcc_target targetm = TARGET_INITIALIZER;
```

그럼 TARGET_INITIALIZER 는 어떻게 구성되어 있을까? 아래와 같다.

```
#define TARGET_INITIALIZER                      \
{                                               \
  TARGET_ASM_OUT,                               \
  TARGET_SCHED,                                 \
  TARGET_MERGE_DECL_ATTRIBUTES,                 \
  TARGET_MERGE_TYPE_ATTRIBUTES,                 \
  TARGET_ATTRIBUTE_TABLE,                       \
  TARGET_COMP_TYPE_ATTRIBUTES,                  \
  TARGET_SET_DEFAULT_TYPE_ATTRIBUTES,           \
  TARGET_INSERT_ATTRIBUTES,                     \
```

```
  TARGET_FUNCTION_ATTRIBUTE_INLINABLE_P,        \
  TARGET_MS_BITFIELD_LAYOUT_P,                  \
  TARGET_INIT_BUILTINS,                         \
  TARGET_EXPAND_BUILTIN,                        \
  TARGET_SECTION_TYPE_FLAGS,                    \
  TARGET_HAVE_NAMED_SECTIONS,                   \
  TARGET_HAVE_CTORS_DTORS,                      \
  TARGET_CANNOT_MODIFY_JUMPS_P                  \
}
```

우선 위의 매크로에 대해서 다시 세부적으로 살펴보기 이전에 기본값들로 어떤 값들이 설정되는지 부터 알아보도록 하자.

4.1 TARGET_ASM_OUT

이 매크로는 struct gcc_target 내에 표현되어 있는 struct asm_out 구조체를 초기화하기 위해서 사용되는 것입니다. 원형은 아래와 같이 선언되어 있습니다.

```
#define TARGET_ASM_OUT                          \
{                                               \
  TARGET_ASM_OPEN_PAREN,                        \
  TARGET_ASM_CLOSE_PAREN,                       \
  TARGET_ASM_BYTE_OP,                           \
  TARGET_ASM_ALIGNED_INT_OP,                    \
  TARGET_ASM_UNALIGNED_INT_OP,                  \
  TARGET_ASM_INTEGER,                           \
  TARGET_ASM_FUNCTION_PROLOGUE,                 \
  TARGET_ASM_FUNCTION_END_PROLOGUE,             \
  TARGET_ASM_FUNCTION_BEGIN_EPILOGUE,           \
  TARGET_ASM_FUNCTION_EPILOGUE,                 \
  TARGET_ASM_NAMED_SECTION,                     \
  TARGET_ASM_EXCEPTION_SECTION,                 \
  TARGET_ASM_EH_FRAME_SECTION,                  \
  TARGET_ASM_CONSTRUCTOR,                       \
  TARGET_ASM_DESTRUCTOR                         \
}
```

이 매크로 또한 다른 하위 매크로들을 많이 포함하고 있다는 사실을 눈으로 확인할 수 있을 것입니다.

여기서 언급되어 있는 매크로에 대해서 계속 알아 보도록 합시다. 처음 나와있는 세 항목에 대한 정보입니다.

```
#define TARGET_ASM_OPEN_PAREN "("
#define TARGET_ASM_CLOSE_PAREN ")"
#define TARGET_ASM_BYTE_OP "\t.byte\t"
```

그리고 네번째와 다섯번째 항목에 대한 매크로 정보입니다.

```
#define TARGET_ASM_ALIGNED_INT_OP               \
{                                               \
        TARGET_ASM_ALIGNED_HI_OP,               \
        TARGET_ASM_ALIGNED_SI_OP,               \
        TARGET_ASM_ALIGNED_DI_OP,               \
        TARGET_ASM_ALIGNED_TI_OP                \
}
```

```
#define TARGET_ASM_UNALIGNED_INT_OP             \
{                                               \
        TARGET_ASM_UNALIGNED_HI_OP,             \
        TARGET_ASM_UNALIGNED_SI_OP,             \
        TARGET_ASM_UNALIGNED_DI_OP,             \
        TARGET_ASM_UNALIGNED_TI_OP              \
}
```

위 매크로 TARGET_ASM_ALIGNED_INT_OP 와
TARGET_ASM_UNALIGNED_INT_OP 는 하위 매크로를 또 포함하고 있는데, 이에 대한 정보는 아래에 나와 있습니다.

```
#define TARGET_ASM_ALIGNED_HI_OP "\t.short\t"
#define TARGET_ASM_ALIGNED_SI_OP "\t.long\t"
#define TARGET_ASM_ALIGNED_DI_OP NULL
#define TARGET_ASM_ALIGNED_TI_OP NULL

/* GAS 와 SYSV4 어셈블러는 아래값들을 받아들입니다.  */
#if defined (OBJECT_FORMAT_ELF) || defined (OBJECT_FORMAT_ROSE)
#define TARGET_ASM_UNALIGNED_HI_OP "\t.2byte\t"
#define TARGET_ASM_UNALIGNED_SI_OP "\t.4byte\t"
#define TARGET_ASM_UNALIGNED_DI_OP "\t.8byte\t"
#define TARGET_ASM_UNALIGNED_TI_OP NULL
#else
#define TARGET_ASM_UNALIGNED_HI_OP NULL
#define TARGET_ASM_UNALIGNED_SI_OP NULL
#define TARGET_ASM_UNALIGNED_DI_OP NULL
#define TARGET_ASM_UNALIGNED_TI_OP NULL
#endif /* OBJECT_FORMAT_ELF || OBJECT_FORMAT_ROSE */
```

여기서 OBJECT_FORMAT_ELF 매크로는
$prefix/gcc/config/elfos.h 파일에 선언되어 있으며 OBJECT_FORMAT_ROSE 매크로는
$prefix/gcc/config/i386/osfrose.h 파일에 선언되어 있습니다.
아래에는 나머지 함수 포인터에 대한 매크로 정보를 가지고 있습니다.

```
#define TARGET_ASM_INTEGER default_assemble_integer

#define TARGET_ASM_FUNCTION_PROLOGUE default_function_pro_epilogue
#define TARGET_ASM_FUNCTION_EPILOGUE default_function_pro_epilogue
#define TARGET_ASM_FUNCTION_END_PROLOGUE no_asm_to_stream
#define TARGET_ASM_FUNCTION_BEGIN_EPILOGUE no_asm_to_stream
```

마지막 항목인 TARGET_ASM_CONSTRUCTOR 와
TARGET_ASM_DESTRUCTOR 를 설정하는 부분입니다.

```
#if !defined(TARGET_ASM_CONSTRUCTOR) && !defined(USE_COLLECT2)
# ifdef CTORS_SECTION_ASM_OP
#  define TARGET_ASM_CONSTRUCTOR default_ctor_section_asm_out_constructor
# else
#  ifdef TARGET_ASM_NAMED_SECTION
#   define TARGET_ASM_CONSTRUCTOR       \
                                default_named_section_asm_out_constructor
#  else
#   define TARGET_ASM_CONSTRUCTOR default_stabs_asm_out_constructor
```

```
#  endif
# endif
#endif

#if !defined(TARGET_ASM_DESTRUCTOR) && !defined(USE_COLLECT2)
# ifdef DTORS_SECTION_ASM_OP
#  define TARGET_ASM_DESTRUCTOR default_dtor_section_asm_out_destructor
# else
#  ifdef TARGET_ASM_NAMED_SECTION
#   define TARGET_ASM_DESTRUCTOR default_named_section_asm_out_destructor
#  else
#   define TARGET_ASM_DESTRUCTOR default_stabs_asm_out_destructor
#  endif
# endif
#endif
```

위의 매크로에서 언급된 USE_COLLECT2 와 CTORS_SECTION_ASM_OP 와 DTORS_SECTION_ASM_OP 는 i386 환경에서는 정의되지 않는 매크로이니 크게 신경을 쓸 필요는 없습니다.

4.2 TARGET_SCHED

이 매크로 또한 위와 마찬가지로 다른 하위 매크로를 포함하고 있는 형태로 구성되어 있습니다. 원형은 아래와 같습니다.

```
#define TARGET_SCHED                            \
{                                               \
  TARGET_SCHED_ADJUST_COST,                     \
  TARGET_SCHED_ADJUST_PRIORITY,                 \
  TARGET_SCHED_ISSUE_RATE,                      \
  TARGET_SCHED_VARIABLE_ISSUE,                  \
  TARGET_SCHED_INIT,                            \
  TARGET_SCHED_FINISH,                          \
  TARGET_SCHED_REORDER,                         \
  TARGET_SCHED_REORDER2,                        \
  TARGET_SCHED_CYCLE_DISPLAY                    \
}
```

내부에 포함되어 있는 각 매크로에 대해서 보도록 해보자.

```
#define TARGET_SCHED_ADJUST_COST 0
#define TARGET_SCHED_ADJUST_PRIORITY 0
#define TARGET_SCHED_ISSUE_RATE 0
#define TARGET_SCHED_VARIABLE_ISSUE 0
#define TARGET_SCHED_INIT 0
#define TARGET_SCHED_FINISH 0
#define TARGET_SCHED_REORDER 0
#define TARGET_SCHED_REORDER2 0
#define TARGET_SCHED_CYCLE_DISPLAY 0
```

위의 모든 것들을 null 포인터로 설정합니다. 이 값들은 i386.c 파일에서 targetm 을 TARGET_INITIALIZER 로 초기화하기 전에 설정됩니다.

4.3 TARGET_MERGE_DECL_ATTRIBUTES

이 매크로는 기본값으로

```
#define TARGET_MERGE_DECL_ATTRIBUTES merge_decl_attributes
```

와 같이 설정됩니다.

4.4 TARGET_MERGE_TYPE_ATTRIBUTES

이 매크로는 기본값으로

```
#define TARGET_MERGE_TYPE_ATTRIBUTES merge_type_attributes
```

와 같이 설정됩니다.

4.5 TARGET_ATTRIBUTE_TABLE

이 매크로는 기본값으로

```
#define TARGET_ATTRIBUTE_TABLE default_target_attribute_table
```

와 같이 설정됩니다.

4.6 TARGET_COMP_TYPE_ATTRIBUTES

이 매크로는 기본값으로

```
#define TARGET_COMP_TYPE_ATTRIBUTES default_comp_type_attributes
```

와 같이 설정됩니다.

4.7 TARGET_SET_DEFAULT_TYPE_ATTRIBUTES

이 매크로는 기본값으로

```
#define TARGET_SET_DEFAULT_TYPE_ATTRIBUTES      \
                        default_set_default_type_attributes
```

와 같이 설정됩니다.

4.8 TARGET_INSERT_ATTRIBUTES

이 매크로는 기본값으로

```
#define TARGET_INSERT_ATTRIBUTES default_insert_attributes
```

와 같이 설정됩니다.

4.9 TARGET_FUNCTION_ATTRIBUTE_INLINABLE_P

이 매크로는 기본값으로

```
#define TARGET_FUNCTION_ATTRIBUTE_INLINABLE_P   \
                        default_function_attribute_inlinable_p
```

와 같이 설정됩니다.

4.10 TARGET_MS_BITFIELD_LAYOUT_P

이 매크로는 기본값으로

```
#define TARGET_MS_BITFIELD_LAYOUT_P default_ms_bitfield_layout_p
```

와 같이 설정됩니다.

4.11 TARGET_INIT_BUILTINS

이 매크로는 기본값으로

```
#define TARGET_INIT_BUILTINS default_init_builtins
```

와 같이 설정됩니다.

4.12 TARGET_EXPAND_BUILTIN

이 매크로는 기본값으로

```
#define TARGET_EXPAND_BUILTIN default_expand_builtin
```

와 같이 설정됩니다.

4.13 TARGET_SECTION_TYPE_FLAGS

이 매크로는 기본값으로

```
#ifndef TARGET_SECTION_TYPE_FLAGS
#define TARGET_SECTION_TYPE_FLAGS default_section_type_flags
#endif
```

와 같이 설정됩니다.

4.14 TARGET_HAVE_NAMED_SECTIONS

이 매크로는 TARGET_ASM_NAMED_SECTION 매크로가 정의되어 있느냐에 따라 값이 달라 질 수 있습니다. 원형은 아래와 같습니다.

```
#ifdef TARGET_ASM_NAMED_SECTION
#define TARGET_HAVE_NAMED_SECTIONS true
#else
#define TARGET_ASM_NAMED_SECTION default_no_named_section
#define TARGET_HAVE_NAMED_SECTIONS false
#endif
```

4.15 TARGET_HAVE_CTORS_DTORS

이 매크로는 아래와 같이 선언되어 있습니다.

```
#if defined(TARGET_ASM_CONSTRUCTOR) && defined(TARGET_ASM_DESTRUCTOR)
#define TARGET_HAVE_CTORS_DTORS true
#else
#define TARGET_HAVE_CTORS_DTORS false
#define TARGET_ASM_CONSTRUCTOR NULL
#define TARGET_ASM_DESTRUCTOR NULL
#endif
```

4.16 TARGET_CANNOT_MODIFY_JUMPS_P

이 매크로는 기본값으로

```
#define TARGET_CANNOT_MODIFY_JUMPS_P hook_void_bool_false
```

와 같이 설정됩니다.

제 5 절 i386.c 에서의 TARGET_INITIALIZER

그럼 이제 i386.c 파일에서는 어떻게 TARGET_INITIALIZER 를 구성할 것인가? 실제로 내용을 보면 아래와 같이 선언되어 있음을 알 수 있습니다.

```
#undef TARGET_ATTRIBUTE_TABLE
#define TARGET_ATTRIBUTE_TABLE ix86_attribute_table
#ifdef TARGET_DLLIMPORT_DECL_ATTRIBUTES
#  undef TARGET_MERGE_DECL_ATTRIBUTES
#  define TARGET_MERGE_DECL_ATTRIBUTES merge_dllimport_decl_attributes
#endif
```

TARGET_ATTRIBUTE_TABLE 매크로를 ix86_attribute_table 함수로 선언합니다.
TARGET_DLLIMPORT_DECL_ATTRIBUTES 는 cygwin 환경에서 gcc 를 컴파일할 때 선언되는 것이기 때문에 저의 환경에서는 선언되지 않습니다.

```
#undef TARGET_COMP_TYPE_ATTRIBUTES
#define TARGET_COMP_TYPE_ATTRIBUTES ix86_comp_type_attributes

#undef TARGET_INIT_BUILTINS
#define TARGET_INIT_BUILTINS ix86_init_builtins

#undef TARGET_EXPAND_BUILTIN
#define TARGET_EXPAND_BUILTIN ix86_expand_builtin

#if defined (OSF_OS) || defined (TARGET_OSF1ELF)
   static void
   ix86_osf_output_function_prologue PARAMS ((FILE *,
                                               HOST_WIDE_INT));
#  undef TARGET_ASM_FUNCTION_PROLOGUE
#  define TARGET_ASM_FUNCTION_PROLOGUE ix86_osf_output_function_prologue
#endif
```

저의 환경에서는 OSF_OS 나 TARGET_OSF1ELF 는 선언되지 않습니다. 그 외에도 아래와 같은 많은 매크로들이 i386/linux 를 위해 맞추어 집니다.

```
#undef TARGET_ASM_OPEN_PAREN
#define TARGET_ASM_OPEN_PAREN ""
#undef TARGET_ASM_CLOSE_PAREN
#define TARGET_ASM_CLOSE_PAREN ""

#undef TARGET_ASM_ALIGNED_HI_OP
#define TARGET_ASM_ALIGNED_HI_OP ASM_SHORT
#undef TARGET_ASM_ALIGNED_SI_OP
#define TARGET_ASM_ALIGNED_SI_OP ASM_LONG
#ifdef ASM_QUAD
#undef TARGET_ASM_ALIGNED_DI_OP
#define TARGET_ASM_ALIGNED_DI_OP ASM_QUAD
#endif

#undef TARGET_ASM_UNALIGNED_HI_OP
#define TARGET_ASM_UNALIGNED_HI_OP TARGET_ASM_ALIGNED_HI_OP
#undef TARGET_ASM_UNALIGNED_SI_OP
#define TARGET_ASM_UNALIGNED_SI_OP TARGET_ASM_ALIGNED_SI_OP
#undef TARGET_ASM_UNALIGNED_DI_OP
#define TARGET_ASM_UNALIGNED_DI_OP TARGET_ASM_ALIGNED_DI_OP

#undef TARGET_SCHED_ADJUST_COST
```

```
#define TARGET_SCHED_ADJUST_COST ix86_adjust_cost
#undef TARGET_SCHED_ISSUE_RATE
#define TARGET_SCHED_ISSUE_RATE ix86_issue_rate
#undef TARGET_SCHED_VARIABLE_ISSUE
#define TARGET_SCHED_VARIABLE_ISSUE ix86_variable_issue
#undef TARGET_SCHED_INIT
#define TARGET_SCHED_INIT ix86_sched_init
#undef TARGET_SCHED_REORDER
#define TARGET_SCHED_REORDER ix86_sched_reorder
```

그럼 저의 환경에서 최종적으로는 어떻게 선언되는 것일까요? 바로 아래와 같이 선언됩니다. 저의 환경에서의 모습입니다.

```
struct gcc_target targetm =
{
  {
    "",
    "",
    "\t.byte\t",
    {
      "\t.value\t",
      "\t.long\t",
      "\t.quad\t",
      NULL
    },
    {
      "\t.value\t",
      "\t.long\t",
      "\t.quad\t",
      NULL
    },
    default_assemble_integer,
    default_function_pro_epilogue,
    no_asm_to_stream,
    no_asm_to_stream,
    default_function_pro_epilogue,
    default_elf_asm_named_section,
    default_exception_section,
    default_eh_frame_section,
    default_named_section_asm_out_constructor,
    default_named_section_asm_out_destructor
  },
  {
    ix86_adjust_cost,
    0,
    ix86_issue_rate,
    ix86_variable_issue,
    ix86_sched_init,
    0,
    ix86_sched_reorder,
    0,
    0
  },
```

```
  merge_decl_attributes,
  merge_type_attributes,
  ix86_attribute_table,
  ix86_comp_type_attributes,
  default_set_default_type_attributes,
  default_insert_attributes,
  default_function_attribute_inlinable_p,
  default_ms_bitfield_layout_p,
  ix86_init_builtins,
  ix86_expand_builtin,
  default_section_type_flags,
  1,
  1,
  hook_void_bool_false
};
```

이제 이 구조체가 어떻게 설정되는지에 대해 살펴보았습니다. 하지만 이 구조체에서 선언된 함수에 대해서 내부적으로 어떻게 행동하는지에 대해서는 이번에는 언급하지 않겠습니다. 나중에 계속 강의를 진행하면서 이에 대한 설명이 필요한 상황이 되면 따로 글을 마련하도록 하겠습니다.

제 6 절 9 주차 강의를 마치며

TARGET_INITIALIZER 구조체에 대해서는 대부분 설명을 마쳤다고 봅니다. 하지만 아직 부족한 부분이 많이 있지만 그 부분에 대해서는 점점 살펴보도록 하겠습니다.

재미있게 즐겨주시기 바랍니다. 행복하세요.