Go to the first, previous, next, last section, table of contents.

어셈블러 지시자들

모든 어셈블러 지시자들은 `.'로 시작된다. 나머지 글자들은 보통 소문자이다.

이 장에서는 GNU 어셈블러가 특정한 기계에 맞게 구성되었는지에는 상관없이 사용 가능한 지시자들을 살펴볼 것이다. 어떤 기계에서의 구성은 추가적인 지시자들을 제공 하기도 한다. See section 기계에 의존하는 것들을 참조하라.

.abort

이 지시자는 어셈블리 작업을 즉각 중단시킨다. 이것은 다른 어셈블러들과 호환 된다. 원래의 생각은 어셈블리 소스가 어셈블러에게 명령을 내리고자 하는 데 있 었다. 만일 소스 제공자가 종결한다면 그것은 as에게 이 지시자를 사용하여 종료 할 수도 있다. 언젠가는 .abort는 제공되지 않을 것이다.

.ABORT

COFF 출력을 생성할 때, as는 `.abort'와 같이 취급한다. `b.out'을 출력할 때는, 이 지시자를 받아들이지만, 무시해 버린다.

When producing b.out output, as accepts this directive, but ignores it.

.align *abs - expr , abs - expr*

현재의 보조섹션에서의 위치 카운터를 어떤 경계선에 정렬시킨다. 첫 번째 표현(절대적이어야 하는)은 위치 카운터가 진척된 후에 가질 작은 순으로 정렬된 숫자이다. 예를 들면 '.align 3'은 위치카운터를 8의 배수가 될 때까지 진척시킬 것이다. 만일 위치 카운터가 8의 배수이면 아무런 변화도 필요치 않다. 두 번째 표현(역시 절대적)은 보태지는 바이트에서 저장될 값을 의미한다. 그것은 (콤마와 함께) 없을 수도 있다. 만일 그것이 없다면, 보태지는 바이트는 0이 될 것이다. For the alpha, if the section is marked as containing code and the padding expression is omitted, then the space is filled with no - ops.

The way the required alignment is specified varies from system to system. For the a29k, hppa, m68k, m88k, w65, sparc, and Hitachi SH, and i386 using ELF format, the first expression is the alignment request in bytes. For example `.align 8' advances

the location counter until it is a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

For other systems, including the i386 using a.out format, it is the number of low-order zero bits the location counter must have after advancement. For example `.align 3' advances the location counter until it a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

This inconsistency is due to the different behaviors of the various native assemblers for these systems which GAS must emulate. GAS also provides .balign and .p2align directives, described later, which have a consistent behavior across all architectures (but are specific to GAS).

.app-file *string*

.app-file ('.file'과 같이 할 수도 있다) 은 as에게 새로운 파일을 시작하고 있음을 알린다. 일반적으로, 이 파일이름은 `""에 둘러싸여진다. 만일 파일이름 을 명시하지 않는다면, ""로 사용할 수도 있다. 이 구문은 앞으로는 사라질 것 이다. 오래된 as 프로그램들과의 호환성 측면에서만 제공된다.

.ascii "*string*"...

.ascii 는 콤마에 의해 구분된 0보다 많은 글자들을 기대한다(see section 문자열). 이것은 각각의 문자열을 연속된 주소로 어셈블한다. 각각의 문자열의 끝 에는 0의 값을 가지는 하나의 바이트가 따라오지 않는다.

.asciz "*string*"...

.asciz 은 .ascii와 같으나 각각의 문자열에는 0의 값을 가지는 한바이트가 따 라온다. .asciz의 "z"은 "zero"를 의미한다.

.balign[wl] *abs-expr* , *abs-expr*

Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the alignment request in bytes. For example `.balign 8' advances the location counter until it is a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

The second expression (also absolute) gives the value to be stored in the padding bytes. It (and the comma) may be omitted. If it is omitted, the padding bytes are

zero.

The .balignw and .balignl directives are variants of the .balign directive. The .balignw directive treats the fill pattern as a two byte word value. The .balignl directives treats the fill pattern as a four byte longword value. For example, .balignw 4,0x368d will align to a multiple of 4. If it skips two bytes, they will be filled in with the value 0x368d (the exact placement of the bytes depends upon the endianness of the processor). If it skips 1 or 3 bytes, the fill value is undefined.

.byte *expressions*

.byte 는 콤마에 의해 구분되는 0개 이상의 표현을 기대한다. 각각의 표현은 다음의 바이트로 어셈블된다.

.comm *symbol , length*

.comm은 bss 섹션의 공통되는 영역을 선언한다. 보통은 ld는 링크 작업을 할 동안 .comm을 위해 메모리 주소를 예약하는 데, 부분적이 아닌 프로그램은 심볼 의 위치를 정의한다. .comm 을 사용하면 ld 에게 적어도 *length* 만큼의 바이트가 필요함을 알릴 수 있다. ld 는 각각의 . comm 심볼을 위해서 적어도 링크된 부분적 인 프로그램들의 어디에서나 .comm 이 요구한 크기 만큼을 할당할 것이다. *length* 는 절대적인 표현이다.

The syntax for .comm differs slightly on the HPPA. The syntax is `*symbol* .comm, *length*'; *symbol* is optional.

.data *subsection*

.data 는 as 에게 따라오는 구문을 *subsection* (절대적 표현)으로 번호가 붙여진 data subsection의 끝부분에 어셈블 하도록 한다. 만일 *subsection* 이 없다면, 그 것은 기본적으로 0이 될 것이다.

.def *name*

심볼 *name* 을 위해서 디버깅 정보를 정의한다. 그 정의는 .endef 지시자가 나올 때까지 확장된다. 이 지시자는 as 가 COFF 포맷 출력으로 설정되었다면 단지 감지만 된다. b.out 을 만들어 낼 때는 `.def'는 인식은 되나 무시된다.

.desc *symbol, abs - expression*

이 지시자는 심볼의 기술자를 absolute expression 의 하위 16비트로 설정한다. (see section 심볼의 속성)

이 `.desc' 지시자는 as 가 COFF 출력으로 설정되었다면 유효하지 않다. 이것은 단지 a.out 이나 b.out 목적 포맷일 때에만 유효하다. 호환성을 위해서, as는 그 것을 허용하나 COFF 설정일 때에는 아무런 출력도 생성하지 않을 것이다.

.dim

이 지시자는 심볼테이블에 추가적인 디버깅 정보를 포함하도록 컴파일러에게 지시한다. It is only permitted inside .def/.endef pairs.

.dim 은 COFF 포맷을 출력할 때에만 의미가 있다. as 가 b.out 을 생성할 때 에는 이 지시자를 받아들이기는 하나 무시해 버린다.

.double *flonums*

.double 은 콤마로 구분되는 0개 이상의 flonums을 기대한다. 그것은 부동 소숫 점 숫자로 어셈블된다. 정확한 부동소숫점 숫자의 종류는 as 가 어떻게 설정되어 있느냐에 따른다. See section 기계에 의존하는 것들.

.eject

어셈블리 리스트를 출력할 때 이 싯점에서 페이지를 넘긴다.

.else

.else 는 as 가 제공하는 조건적인 어셈블리의 한부분이다. see section *.if absolute expression*를 참조하라. 그것은 .if 에 따르는 조건이 거짓일 경우 어셈블될 코드의 시작부분을 표시한다.

.endef

이 지시자는 .def 로 정의된 심볼의 끝을 표시한다.

`.endif'는 COFF 포맷 출력시에만 의미가 있다. as가 b.out 을 생성하도록 설 정되었다면 이 지시자를 받아들이기는 하나 무시된다.

.endif

.endif 는 as 가 제공하는 조건적 어셈블리의 일부분이다. 이것은 조건적으로 어셈블된 코드의 블록의 마지막을 표시한다. See section *.if absolute expression*.

.equ *symbol, expression*

이 지시자는 *symbol*의 값을 *expression*으로 설정한다. 이것은 `.set'과 같은 뜻 이다; see section *.set symbol, expression*.

The syntax for equ on the HPPA is `*symbol* .equ *expression*'.

.extern

.extern 은 다른 어셈블러들과의 호환성을 위해서 소스 프로그램에 나타나는 것을 허용한다. 그러나 이것은 무시된다. as는 모든 정의되지 않은 심볼을 external 로 취급한다.

.file *string*

.file (`.app-file'로 불리워지기도 한다.) 은 새로운 논리적인 파일의 시작임 을 as에게 알린다. string은 새로운 파일의 이름이다. 일반적으로, 이 파일이름은 `""'로 둘러싸인다. 그러나 파일이름을 명시하지 않으려면 `""'"를 사용하여야 한 다. 이 구문은 앞으로는 사라질 것이다. 이것은 단지 오래된 as 프로그램과의 호환 성을 위한 것이다. 어떤 as의 구성에서는 .file은 다른 어셈블러들과의 충돌을 피 하기 위해서 제거되었다. See section 기계에 의존하는 것들.

.fill *repeat , size , value*

result, size, value 는 절대적인 표현이다. 이것은 *size* 바이트만큼 복사를 반 복한다. *Repeat* 은 0이상이 될 것이다. *Size*는 0이상이 될 수 있으나 다른 어셈 블러들과의 호환성을 위해서 8 이상일 것이다. 각각의 *repeat* 바이트들의 구성은 8 바이트 숫자로부터 취해진다. 상위 4바이트는 0이다. 하위 4바이트는 as가 어셈블 링하는 컴퓨터상의 정수의 바이트순서로 되는 *value* 이다. 한 번의 반복에서 각각 의 size 바이트는 이 숫자의 낮은 순으로 정렬된 *size* 바이트로 부터 취한다. 다시 말하지만 이 기괴한 행동은 다른 사람들의 어셈블러들과 호환성을 가지게 하기 위한 것이다.

size 와 *value* 는 선택사항이다. 만일 두 번째 콤마와 *value*가 없다면, *value*는 0 으로 가정된다. 만일 첫 번째 콤마와 따라오는 토큰이 없다면 *size* 는 1로 가정된다.

.float *flonums*

이 지시자는 콤마에 의해 구분되는 0개 이상의 flonums를 어셈블한다. 이 것은 .single과 같은 뜻이다. 정확한 부동소숫점 숫자의 종류는 as가 어떻게 설정 되었느냐에 따른다. See section 기계에 의존하는 것들.

.global *symbol*, .globl *symbol*

.global 은 ld 에게 보이는 심볼을 만든다. 만일 여러분이 *symbol*을 여러분의 부분적인 프로그램에 만든다면, 그것의 값은 그것이 링크되는 다른 부분적인 프 로그램들에게도 유효하게 된다. 그렇지 않다면, *symbol* 은 그것이 링크되는 다른 부분적인 프로그램으로 부터 같은 이름의 심볼의 속성을 가질 것이다.

`.globl'과 `.global' 둘 다 다른 어셈블러들과의 호환성을 위해서 허용된다. On the HPPA, .global is not always enough to make it accessible to other partial programs. You may need the HPPA - only .EXPORT directive as well. See section HPPA Assembler Directives.

.hword *expressions*

이것은 0개이상의 *expressions* 를 기대한다. 그리고 각각은 16 비트 숫자이다.

이 지시자는 해당 기계에 의존하는 `.short'와 같은 뜻이며, 또한 `.word'와 도 같은 뜻일 것이다.

.ident

이 지시자는 어떤 어셈블러들이 목적파일에 태그를 설치하는 데에 사용된다. as 는 다른 어셈블러들과의 소스파일 호환성을 위해서 그러한 지시자를 허용한 다. 그러나 그것을 위해 별다른 것이 활성화되지는 않을 것이다.

.if *absolute expression*

.if 는 만일 인자(*absolute expression*이어야 하는)가 0이 아니라면 어셈블되는 소스 프로그램의 일부분으로 구성되는 코드영역의 시작을 표시한다. 그 조건적인 코 드영역의 끝은 .endif 에 의해 마크된다. (see section .endif). 선택적 으로 .else 로 표시되는 선택적인 조건을 위한 코드를 포함시킬 수도 있다. (see section .else)

아래와 같은 .if 의 다른 변형들도 제공된다.

.ifdef *symbol*
특정한 *symbol*이 정의되었을 경우에 그 뒤의 코드 영역을 어셈블한다.
.ifndef *symbol*
.ifnotdef *symbol*
특정한 *symbol*이 정의되지 않았을 경우에 그 뒤의 코드 영역을 어셈블한다. 둘 다 같은 뜻이다.

.include "*file*"

이 지시자는 소스 프로그램에서 명시된 위치에 제공되는 파일을 포함하는 방법 을 제공한다. *file* 에 있는 코드는 .include 의 위치에 따라오는 것처럼 어셈블 되며, 포함되는 파일의 끝이 나타나면 원래 파일의 어셈블은 다시 계속된다. `-I' 명령행 옵션 (see section 명령행)을 사용하여 검색경로를 제 어할 수 있다. *file* 앞뒤로 인용부호가 필요하다.

.int *expressions*

콤마에 의해 구분되는 어떤 섹션의 0개 이상의 *expressions*을 적을 수 있다. 각각의 표현은 실행시에 그 표현의 값이 될 32비트 숫자가 될 것이다. expression에서 의 바이트 순서는 프로그램이 돌아가는 컴퓨터의 종류에 따라 다르다.

.irp *symbol,values...*

Evaluate a sequence of statements assigning different values to *symbol*. The sequence of statements starts at the .irp directive, and is terminated by an .endr directive. For each *value*, *symbol* is set to *value*, and the sequence of statements is assembled. If no *value* is listed, the sequence of statements is assembled once, with *symbol* set to the null string. To refer to *symbol* within the sequence of statements, use ₩*symbol*.

For example, assembling

```
.irp    param,1,2,3
move    d₩param,sp@ -
.endr
```

is equivalent to assembling

```
move    d1,sp@ -
```

```
        move    d2,sp@ -
        move    d3,sp@ -
```

.irpc *symbol,values...*

Evaluate a sequence of statements assigning different values to *symbol*. The sequence of statements starts at the .irpc directive, and is terminated by an .endr directive. For each character in *value*, *symbol* is set to the character, and the sequence of statements is assembled. If no *value* is listed, the sequence of statements is assembled once, with *symbol* set to the null string. To refer to *symbol* within the sequence of statements, use ₩*symbol*.

For example, assembling

```
        .irpc   param,123
        move    d₩param,sp@ -
        .endr
```

is equivalent to assembling

```
        move    d1,sp@ -
        move    d2,sp@ -
        move    d3,sp@ -
```

.lcomm *symbol , length*

symbol 에 의해 표시되는 지역적으로 공통되는 것들을 위해서 *length* (절대적 표현) 바이트만큼을 예약한다. *symbol*의 섹션과 값은 새로운 지역적인 공통되는 것들이다. 그 주소는 bss 섹션에 할당되고, 실행시 그 바이트는 0부터 시작될 것 이다. *Symbol* 은 global로 선언되지 않는다. (see section .global *symbol*, .globl *symbol*) 그래서 *Symbol* 은 ld 에게는 보이지 않는다.

The syntax for .lcomm differs slightly on the HPPA. The syntax is `*symbol* .lcomm, *length*'; *symbol* is optional.

.lflags

as 는 다른 어셈블러들과의 호환성 차원에서 허용하나 그것을 곧바로 무시해 버린다.

.line *line - number*

as 에게 논리적인 줄 번호를 바꾸도록 지시한다. *line - number* 은 절대적 표현 이어야 한다. 그 다음 줄은 논리적인 줄 번호를 가질 것이다. 그래서 현재의 줄 (구분자 구문 이후에)에서 다른 어떤 구문들도 논리적인 줄번호 *line - number* - 1 로 보고 될 것이다. 이 지시자도 언젠가는 제공되지 않을 것이다. 이것은 단지 다른 어셈블러 프로그램들과의 호환성 측면에서 사용된다.

경고: as 의 AMD29K 설정에서는 이러한 명령은 단지 .line 이나 .ln 중 .ln 만이 유효하다.

이것은 a.out 과 b.out 목적코드 포맷과도 관계되는 지시자이며, as 는 COFF 출력을 생성할 때에도 그것을 인식하여, 그것이 .def/.endef 짝의 외곽에 있다면 `.line'을 COFF `.ln'으로 취급할 것이다.

.def 안쪽의 `.line' 은 대신, 디버깅을 위한 추가적인 심볼 정보를 생성하는 데에 컴파일러에 의해 사용되는 지시자의 하나이다.

.linkonce [*type*]

Mark the current section so that the linker only includes a single copy of it. This may be used to include the same section in several different object files, but ensure that the linker will only include it once in the final output file. The .linkonce pseudo - op must be used for each instance of the section. Duplicate sections are detected based on the section name, so it should be unique.

This directive is only supported by a few object file formats; as of this writing, the only object file format which supports it is the Portable Executable format used on Windows NT.

The *type* argument is optional. If specified, it must be one of the following strings. For example:

.linkonce same_size

Not all types may be supported on all object file formats.

discard
: Silently discard duplicate sections. This is the default.

one_only
: Warn if there are duplicate sections, but still keep only one copy.

same_size

Warn if any of the duplicates have different sizes.

same_contents

Warn if any of the duplicates do not have exactly the same contents.

.ln *line - number*

`.ln' 은 `.line'과 동의어이다.

.mri *val*

If *val* is non - zero, this tells as to enter MRI mode. If *val* is zero, this tells as to exit MRI mode. This change affects code assembled until the next .mri directive, or until the end of the file. See section Assemble in MRI Compatibility Mode: - M.

.list

.nolist 지시자와 더불어 생성되는 어셈블리 리스팅을 제어한다. 이 두 개의 지시자는 내부적인 카운터(0으로 초기화되는)를 관리한다. .list 는 카운트를 증가시키고, .nolist 는 감소시킨다. 어셈블리 리스팅은 카운터가 0이상일 때에 는 언제든지 생성시킨다.

기본적으로, 리스팅은 하지 않는다. 그것들을 초기화할 때 (` - a' 명령행 옵션 과 관련해서는 See section 명령행.) 그 리스팅 카운터의 초기값은 1이다.

.long *expressions*

.long은 `.int'와 같다. see section .int *expressions*.

.macro

The commands .macro and .endm allow you to define macros that generate assembly output. For example, this definition specifies a macro sum that puts a sequence of numbers into memory:

```
.macro  sum from=0, to=5
.long   ₩from
.if     ₩to - ₩from
sum     "(₩from+1)",₩to
.endif
.endm
```

With that definition, `SUM 0,5' is equivalent to this assembly input:

```
.long   0
.long   1
.long   2
.long   3
.long   4
.long   5
```

.macro *macname*
.macro *macname macargs ...*
Begin the definition of a macro called *macname*. If your macro definition requires arguments, specify their names after the macro name, separated by commas or spaces. You can supply a default value for any macro argument by following the name with `=*deflt*'. For example, these are all valid .macro statements:
.macro comm
Begin the definition of a macro called comm, which takes no arguments.
.macro plus1 p, p1
.macro plus1 p p1
Either statement begins the definition of a macro called plus1, which takes two arguments; within the macro definition, write `₩p' or `₩p1' to evaluate the arguments.
.macro reserve_str p1=0 p2
Begin the definition of a macro called reserve_str, with two arguments. The first argument has a default value, but not the second. After the definition is complete, you can call the macro either as `reserve_str *a*, *b*' (with `₩p1' evaluating to *a* and `₩p2' evaluating to *b*), or as `reserve_str ,*b*' (with `₩p1' evaluating as the default, in this case `0', and `₩p2' evaluating to *b*).
When you call a macro, you can specify the argument values either by position, or by keyword. For example, `sum 9,17' is equivalent to `sum to=17, from=9'.
.endm
Mark the end of a macro definition.
.exitm
Exit early from the current macro definition.
₩@
as maintains a counter of how many macros it has executed in this pseudo-variable; you can copy that number to your output with `₩@', but *only within a macro definition.*

.nolist

.list 지시자와 더불어 생성되는 어셈블리 리스팅을 제어한다. 이 두 개의 지시자는 내부적인 카운터(0으로 초기화되는)를 관리한다. .list는 카운트를 증 가시키고, .nolist 는 감소시킨다. 어셈블리 리스팅은 카운터가 0이상일 때에는 언제든지 생성시킨다.

.octa *bignums*

이 지시자는 콤마로 구분되는 0개이상의 bignums 를 기대한다. 각각의 bignum 은 하나의 16 바이트 정수이다.

The term "octa" comes from contexts in which a "word" is two bytes; hence *octa* - word for 16 bytes.

.org *new - lc , fill*

.org 는 현재 섹션의 위치 카운터를 *new - lc* 로 진전시킬 것이다. *new - lc*는 절대 표현이거나 현재의 보조섹션에서 같은 섹션의 표현이다. 이 말은 여러들은 섹션 을 가로지르기 위해 .org 를 사용할 수 없다는 뜻이다. *new - lc* 가 잘못된 섹션 을 가지고 있다면, .org 지시자는 무시된다. 다른 어셈블러들과의 호환성을 위해 서 *new - lc* 의 섹션의 절대치라면 as는 그것을 현재 보조섹션과 *new - lc*의 섹션을 같다고 취급하는 것보다는 경고를 발행할 것이다.

.org 는 위치 카운터를 증가시키거나 그것을 변경시키지 않을 경우에 사용된다. .org 로 위치 카운터를 거꾸로 되돌릴 수는 없다.

as는 한 번만에 (one pass로) 프로그램을 어셈블하려고 하기 때문에 *new - lc*는 정의될 것이다. 이러한 제한이 혐오스럽다면 우리는 여러분들의 진보된 어셈블러들을 공유할 수 있게 될 기회를 간절히 기다린다.

위치 카운터의 시작은 보조섹션의 시작이 아니라 섹션의 시작과 관련되어 있음 을 조심하라. 이것은 다른 사람들의 어셈블러들과 호환성을 가지기 위한 것이다.

현재의 보조섹션에서 위치 카운터가 전진 될 때, 그 사이의 바이트는 절대표현 인 *fill* 로 채워진다. 만일 콤마와 *fill* 이 생략된다면 *fill*은 기본으로 0이 된다.

.p2align[wl] *abs - expr , abs - expr*

Pad the location counter (in the current subsection) to a particular storage boundary. The first expression (which must be absolute) is the number of low - order zero bits the location counter must have after advancement. For example `.p2align 3' advances

the location counter until it a multiple of 8. If the location counter is already a multiple of 8, no change is needed.

The second expression (also absolute) gives the value to be stored in the padding bytes. It (and the comma) may be omitted. If it is omitted, the padding bytes are zero.

The .p2alignw and .p2alignl directives are variants of the .p2align directive. The .p2alignw directive treats the fill pattern as a two byte word value. The .p2alignl directives treats the fill pattern as a four byte longword value. For example, .p2alignw 2,0x368d will align to a multiple of 4. If it skips two bytes, they will be filled in with the value 0x368d (the exact placement of the bytes depends upon the endianness of the processor). If it skips 1 or 3 bytes, the fill value is undefined.

.psize *lines , columns*

리스트를 생성할 때 줄수를 선언하거나 선택적으로 칸수 (각각의 페이지에서) 를 선언할 때 사용한다.

.psize 를 사용하지 않는 다면 기본은 60줄로 계산된다. 콤마와 *columns*는 생략 할 수 있으며 이것의 기본은 200 칸이다.

as generates formfeeds whenever the specified number of lines is exceeded (or whenever you explicitly request one, using .eject).

lines 를 0으로 명시한다면, 폼피드는 생성되지 않는다.

.quad *bignums*

.quad 는 콤마로 구분되는 0개 이상의 bignums를 기술한다. 각각의 bignum은 8 바이트 정수이다. bignum이 8바이트를 만족시키지 못한다면,

"quad" 는 2바이트인 "word"와 같은 것으로부터 유래되었다. 그러므로 *quad* - word 는 8바이트이다.

.rept *count*

Repeat the sequence of lines between the .rept directive and the next .endr directive *count* times.

For example, assembling

```
.rept   3
.long   0
.endr
```

is equivalent to assembling

```
.long   0
.long   0
.long   0
```

.sbttl "*subheading*"

어셈블리 리스트를 만들어 낼 때 타이틀(타이틀 라인 바로뒤의 세 번째 줄)로 *subheading*을 사용하라.

이 지시자는 페이지의 상단 10줄 안에 나타날 경우 다음에 이어지는 페이지에 도 영향을 미친다.

.scl *class*

심볼을 위한 저장 클래스(storage - class)를 설정한다. 이 지시자는 .def/.endef 의 안에서 사용된다. 저장 클래스는 심볼이 정적이거나 외부적일 경우, 더 나아가 그것이 심볼릭 디버깅 정도까지도 포함할 경우 표시된다.

`.scl' 지시자는 먼저 COFF 출력과 연관된다. b.out 출력 포맷으로 생성하는 것으로 설정되었을 경우에는 as는 이 지시자를 허용하나 무시해버린다.

.section *name, subsection*

COFF에서 *name* 섹션이라 불리우는 *subsection*의 끝부분으로 다음의 코드를 어셈블한다. *subsection*을 생략한다면, as는 보조섹션 0으로 사용한다. `.section .text'는 .text 지시자와 동일하다. `.section .data' 는 .data 지시자와 동일 하다. This directive is only supported for targets that actually support arbitrarily named sections; on a.out targets, for example, it is not accepted, even with a standard a.out section name as its parameter.

.set *symbol, expression*

이 지시자는 *symbol*의 값을 *expression*으로 설정한다. 이것은 *symbol*의 값과 형을 *expression*으로 따르게 한다. 만일 *symbol*이 외부적으로 표시되었다면 그것 은 표시만 남긴다. (See section 심볼의 속성.)

같은 어셈블리 소스에서 여러번 심볼을 .set 할 수 있다.

전역심볼을 .set 한다면 목적파일에 저장되는 값은 그곳에 저장되는 마지막 값이 될 것이다.

The syntax for set on the HPPA is `*symbol* .set *expression*'.

.short *expressions*

.short 는 보통 `.word'와 같다. See section .word *expressions*.

어떤 설정에 있어서는, .short 와 .word는 다른 길이의 숫자들을 생성한다. see section 기계에 의존하는 것들.

.single *flonums*

이 지시자는 콤마로 구분되는 0개 이상의 flonums를 어셈블한다. 이것은 .float 와 같은 영향을 미친다. 정확한 부동소숫점 숫자의 종류는 as가 어떻게 설정되 었는지에 따른다. See section 기계에 의존하는 것들.

.size

이 지시자는 심볼 테이블에 추가적인 디버깅 정보를 포함하도록 어셈블러에게 지시한다. 이것은 .def/.endef 의 사이에 나타나는 것만이 허용된다.

`.size' 는 단지 COFF 포맷을 출력할 때만 의미가 있다. b.out을 출력할 때는 허용은 되나 이 지시자는 무시된다.

.skip *size* , *fill*

This directive emits *size* bytes, each of value *fill*. Both *size* and *fill* are absolute expressions. If the comma and *fill* are omitted, *fill* is assumed to be zero. This is the same as `.space'.

.space *size* , *fill*

이 지시자는 *size* 바이트를 *fill* 의 값으로 채운다. *size* 와 *fill* 은 둘다 절대 적인 표현이어야 한다. 만일 콤마와 *fill* 이 생략되었다면, *fill* 은 0으로 가정 된다. This is the same as `.skip'.

> *경고:* GNU 어셈블러의 거의 대부분의 버전은 .space 지시자는 .block 지시자와 같은 영향을 미친다. .space 지시자에 관해서는 *HP9000 Series 800 Assembly Language Reference Manual* (HP 92432 - 90001)를 보라. 요약은 See section HPPA Assembler Directives.

AMD 29K 상에서는 이 지시자는 무시된다. 다른 AMD29K 어셈블러들과의 호환성 차원에서 허용된다.

> *Warning:* In most versions of the GNU assembler, the directive .space has the effect of .block See section 기계에 의존하는 것들.

.stabd, .stabn, .stabs

`.stab'로 시작하는 세 개의 지시자들은 심볼릭 디버거의 사용목적을 위해서 심볼을 만든다. (see section 심볼) 이 심볼들은 as의 해쉬 테이블에는 들어가지 않는다. 그들은 소스파일이 아닌 다른 어느곳에서는 참조되지 않는다. 5개의 필드 이상이 요구된다.

string
: 이것은 심볼의 이름이다. `₩000'을 제외하고는 어느 문자나 포함할 수 있다. 그래서 이것은 보통의 심볼 이름들보다는 좀더 범용적이다. 어떤 디버거들은 복잡한 구조들을 이러한 필드를 사용한 심볼 이름으 로 코딩한다.

type
: 절대적인 표현. 심볼의 형은 이 표현의 하위 8비트로 세트된다. 어떤 비트 패턴도 허용되나 ld 와 디버거들은 멍청한(silly) 비트 페턴은 막을 것이다.

other
: 절대적인 표현. 심볼의 "other" 속성은 이 표현의 하위 8비트로 세트된다.

desc
: 절대적인 표현. 심볼의 기술자는 이 표현의 하위 16비트로 세트된다.

value
: 절대적인 표현의 심볼의 값.

만일 .stabd, .stabn, .stabs 구문을 읽는 동안 경고가 발생하면, 심볼은 아 마도 이미 만들어 졌을 것이고 목적파일에 반쯤 형성된 심볼을 얻을 것이다. 이 것은 이전의 어셈블러들과의 호환성을 위한 것이다.

.stabd *type* , *other* , *desc*
: 만들어 지는 심볼의 "이름"은 빈 문자열이 아니다. 그것은 호환성을 위해 널 포인터이다. 오래된 어셈블러들은 널 포인터를 사용하는 데 그들은 목적파일에 빈 문자열로 공간을

낭비하지 않는다. 이 심볼의 값은 위치 카운터의 값으로 설정된다. 프로그램이 링크될 때, 심볼의 값은 .stdbd 가 어셈블 될 때의 위치 카운터가 있는 곳이다.

.stabn *type* , *other* , *desc* , *value*

심볼의 이름은 빈 문자열 ""로 세트된다.

.stabs *string* , *type* , *other* , *desc* , *value*

다섯 개의 필드 모두가 명시되었다.

.string "*str*"

Copy the characters in *str* to the object file. You may specify more than one string to copy, separated by commas. Unless otherwise specified for a particular machine, the assembler marks the end of each string with a 0 byte. You can use any of the escape sequences described in section 문자열.

.tag *structname*

이 지시자는 심볼테이블에 추가적인 디버깅 정보를 포함하도록 컴파일러에게 지시한다. .def/.endef 의 안에서만 허용된다. 태그는 심볼 테이블의 구조 정의 들을 링크하는 데 사용된다.

`.tag' 는 단지 COFF 출력 포맷을 생성할 때에만 사용된다. as가 b.out 을 생성할 때에는 그것은 허용되나 무시된다.

.text *subsection*

어셈블러에게 다음의 구문을 *subsection*으로 번호붙여진 text 섹션의 끝으로 어셈블 하도록 지시한다. 이것은 절대적인 표현이며 *subsection* 이 생략된다면, 섹션 숫자 0이 사용된다.

.title "*heading*"

어셈블리 리스트를 생성할 때 제목(소스파일의 이름과 페이지 숫자 바로 뒤의 2번 째 줄)으로 *heading*을 사용한다.

이 지시자는 다음에 계속되는(subsequent) 페이지에 영향을 미친다. 이것이 페이 지의 상단 10줄안에 나타났다면 현재 페이지만큼..

.type *int*

이 지시자는 .def/.endef 안에서만 허용되며 심볼 테이블 엔트리의 형 속성으로 정수 *int*를 기록한다.

`.type'은 COFF 포맷 출력에만 연관되고, b.out 출력일 때에는 이것은 받아는 들여지나 곧이어 무시된다.

.val *addr*

이 지시자는 .def/.endef 안에서만 허용되며 심볼 테이블 엔트리의 값 속성을 주소 addr 로 기록한다.

`.val'은 COFF 출력 일때에만 사용되고, b.out을 출력 할 때에는 이 지시자는 허용되나 무시된다.

.word *expressions*

이 지시자는 콤마에 의해 구분되는 0개 이상의 어떠한 섹션에서의 *expressions* 을 기대한다.

숫자의 크기는 채워지고,그것의 바이트 순서는 어떤 컴퓨터가 프로그램을 실행하고 있는 가에 따라 다르다.

Warning: Special Treatment to support Compilers

32비트 주소 공간의 기계가 32비트 주소지정보다 적을 경우, 다음에 따라오는 특별 한 대우를 요구한다. 당신에게 흥미로운 기계가 32비트 주소지정을 한다면 (또는 그것을 요구하지 않는다면), 이번 문제는 무시해도 괜찮다. see section 기계에 의존하는 것들.

어셈블을 할 때의 순서에 있어서 컴파일러는 작업할 어떤 것속으로 출력하고, as 는 특별하게 이상한 것은 `.word' 지시자로 작업한다. 지시자의 형태 `.word sym1 - sym2' 는 종종 컴파일러에 의해 점프 테이블의 일부분으로 발생한다. 그러 므로, as 가 `.word sym1 - sym2'의 지시자 형태를 어셈블 할 때에, sym1 과 sym2 의 차이점은 16비트로 채워지지 않는다. as 는 다음 라벨 이전에 즉각적으로 *제 2의 점프 테이블*을 만들 것이다. 제2의 점프 테이블은 제2의 테이블 다음의 첫 바이트로의 short - jump 에 의해 앞질러 진다. 이 short - jump 는 새로운 테이블속 으로 우연히 떨어지는 것으로 부터 흐름의 제어를 방해한다. 테이블의 안쪽은 sym2 로의 long - jump가 될 것이다. 원래의 `.word'는 sym1 - (sym2로의 long - jump의 주소) 의 계산값을 포함할 것이다.

만일 여러번의 `.word sym1 - sym2'가 제2의 점프 테이블이전에 나타난다면, 그것 들의 모두는 조절될 것이다. 만일 `.word sym3 - sym4'와 같은 것이 있으면, 이것도 역시 16비트에서 만족시키지 못한다. sym4로의 long - jump 는 제2의 점프 테이블에 서 포함될 것이고, .word 지시자는 sym3 - (sym4로의 long - jump 주소)의 값을 포함하도록 조절될 것이다. 그리고 필요한 만큼의 원래의 점프 테이블에서 여러개 의 엔트리 만큼 계속될 것이다.

숭악한 지시자들

언젠가는 이러한 지시자들은 제공되지 않을 것이다. 이것들은 오래된 어셈블러 들과의 호환성 유지차원에서 제공된다.

.abort
.app - file
.line

Go to the first, previous, next, last section, table of contents.