

# StrongForth.f 3.1 Glossary: assembler

<b>.instruction ( -- )</b>	asm.sf
Send a disassembly of the assembler instruction starting at the disassembler's location counter to the default output stream. Advance the location counter to the next assembler instruction.	
<b>.location ( -- )</b>	asm.sf
Send the disassembler's location counter in an eight-digit hexadecimal format with a trailing colon and a space character to the default output stream.	
<b>.word ( single -- )</b>	asm.sf
Send the 16-bit value <code>single</code> in a four-digit hexadecimal format with no trailing space to the default output stream.	
<b>16-bit-address: ( -- )</b>	asm.sf
Compile the next assembler instruction with a 16-bit address prefix.	
<b>16-bit-operand: ( -- )</b>	asm.sf
Compile the next assembler instruction with a 16-bit operand prefix.	
<b>aaa, ( -- )</b>	asm.sf
Compile an <code>aaa</code> assembler instruction.	
<b>aad, ( -- )</b>	asm.sf
Compile an <code>aad</code> assembler instruction.	
<b>aad, ( unsigned -- )</b>	asm.sf
Compile an <code>aad</code> assembler instruction with <code>unsigned</code> being the number base.	
<b>aam, ( -- )</b>	asm.sf
Compile an <code>aam</code> assembler instruction.	
<b>aam, ( unsigned -- )</b>	asm.sf
Compile an <code>aam</code> assembler instruction with <code>unsigned</code> being the number base.	
<b>aas, ( -- )</b>	asm.sf
Compile an <code>aas</code> assembler instruction.	

**adc, ( mode mode -- )** asm.sf

Compile an `adc` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**adc, ( mode single -- )** asm.sf

Compile an `adc` assembler instruction with `mode` being the destination and `single` being the immediate source operand.

**add, ( mode mode -- )** asm.sf

Compile an `add` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**add, ( mode single -- )** asm.sf

Compile an `add` assembler instruction with `mode` being the destination and `single` being the immediate source operand.

**again, ( destination-address -- )** asm.sf

Compile a `jmp` assembler instruction with `destination-address` being the jump destination.

**ah ( -- mode )** asm.sf

`mode` is the register direct `ah` addressing mode.

**ahead, ( -- origin-address )** asm.sf

Compile a `jmp` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**al ( -- mode )** asm.sf

`mode` is the register direct `al` addressing mode.

**and, ( mode mode -- )** asm.sf

Compile an `and` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**and, ( mode single -- )** asm.sf

Compile an `and` assembler instruction with `mode` being the destination and `single` being the immediate source operand.

**any: ( stack-diagram -- 1st )** asm.sf

When used in a stack diagram, specifies the succeeding input or output parameter shall be assigned to any single register or register pair. An exception is thrown if the input or output parameter does not fit into a single register or a register pair.

**arpl, ( mode mode -- )**

asm.sf

Compile an `arpl` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**ax ( -- mode )**

asm.sf

`mode` is the register direct `ax` addressing mode.

**begin, ( -- destination-address )**

asm.sf

`destination-address` is the next free location of the code space.

**bh ( -- mode )**

asm.sf

`mode` is the register direct `bh` addressing mode.

**bl ( -- mode )**

asm.sf

`mode` is the register direct `bl` addressing mode.

**bound, ( mode mode -- )**

asm.sf

Compile a `bound` assembler instruction with the first `mode` being the array index and the second `mode` being the bounds operand.

**bp ( -- mode )**

asm.sf

`mode` is the register direct `bp` addressing mode.

**bsf, ( mode mode -- )**

asm.sf

Compile a `bsf` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**bsr, ( mode mode -- )**

asm.sf

Compile a `bsr` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**bswap, ( mode -- )**

asm.sf

Compile a `bswap` assembler instruction with `mode` being the destination register.

**bt, ( mode mode -- )**

asm.sf

Compile a `bt` assembler instruction with the first `mode` being the bit string and the second `mode` being the bit offset in a register.

**`bt, ( mode unsigned -- )`**

asm.sf

Compile a `bt` assembler instruction with `mode` being the bit string and `unsigned` being the immediate bit offset.

**`btc, ( mode mode -- )`**

asm.sf

Compile a `btc` assembler instruction with the first `mode` being the bit string and the second `mode` being the bit offset in a register.

**`btc, ( mode unsigned -- )`**

asm.sf

Compile a `btc` assembler instruction with `mode` being the bit string and `unsigned` being the immediate bit offset.

**`btr, ( mode mode -- )`**

asm.sf

Compile a `btr` assembler instruction with the first `mode` being the bit string and the second `mode` being the bit offset in a register.

**`btr, ( mode unsigned -- )`**

asm.sf

Compile a `btr` assembler instruction with `mode` being the bit string and `unsigned` being the immediate bit offset.

**`bts, ( mode mode -- )`**

asm.sf

Compile a `bts` assembler instruction with the first `mode` being the bit string and the second `mode` being the bit offset in a register.

**`bts, ( mode unsigned -- )`**

asm.sf

Compile a `bts` assembler instruction with `mode` being the bit string and `unsigned` being the immediate bit offset.

**`bx ( -- mode )`**

asm.sf

`mode` is the register direct `bx` addressing mode.

**`byte[eax] ( -- mode )`**

asm.sf

`mode` is the register indirect `[eax]` addressing mode for byte operands.

**`byte[eax]+ ( single -- mode )`**

asm.sf

`mode` is the register indirect with displacement `[eax] +` addressing mode for byte operands.  
`single` is the displacement.

**byte[ebp] ( -- mode )** asm.sf

mode is the register indirect [ebp] addressing mode for byte operands.

**byte[ebp]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [ebp] + addressing mode for byte operands.  
single is the displacement.

**byte[ebx] ( -- mode )** asm.sf

mode is the register indirect [ebx] addressing mode for byte operands.

**byte[ebx]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [ebx] + addressing mode for byte operands.  
single is the displacement.

**byte[ecx] ( -- mode )** asm.sf

mode is the register indirect [ecx] addressing mode for byte operands.

**byte[ecx]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [ecx] + addressing mode for byte operands.  
single is the displacement.

**byte[edi] ( -- mode )** asm.sf

mode is the register indirect [edi] addressing mode for byte operands.

**byte[edi]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [edi] + addressing mode for byte operands.  
single is the displacement.

**byte[edx] ( -- mode )** asm.sf

mode is the register indirect [edx] addressing mode for byte operands.

**byte[edx]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [edx] + addressing mode for byte operands.  
single is the displacement.

**byte[esi] ( -- mode )** asm.sf

mode is the register indirect [esi] addressing mode for byte operands.

**byte[esi]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [esi] + addressing mode for byte operands.  
single is the displacement.

**byte[esp] ( -- mode )** asm.sf

mode is the register indirect [esp] addressing mode for byte operands.

**byte[esp]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [esp] + addressing mode for byte operands.  
single is the displacement.

**byte[] ( single -- mode )** asm.sf

mode is the direct addressing mode for byte operands. single is the memory address.

**call, ( address -- )** asm.sf

Compile a call instruction to destination address.

**call, ( mode -- )** asm.sf

Compile a call instruction to destination mode.

**callf, ( mode -- )** asm.sf

Compile a far call instruction to destination mode.

**callf, ( segment address -- )** asm.sf

Compile a far call instruction to the destination specified by segment and address.

**cbw, ( -- )** asm.sf

Compile a cbw assembler instruction.

**cdq, ( -- )** asm.sf

Compile a cdq assembler instruction.

**ch ( -- mode )** asm.sf

mode is the register direct ch addressing mode.

**cl ( -- mode )** asm.sf

mode is the register direct `cl` addressing mode.

**`clc, ( -- )`** asm.sf

Compile a `clc` assembler instruction.

**`cld, ( -- )`** asm.sf

Compile a `cld` assembler instruction.

**`cli, ( -- )`** asm.sf

Compile a `cli` assembler instruction.

**`clts, ( -- )`** asm.sf

Compile a `clts` assembler instruction.

**`cmc, ( -- )`** asm.sf

Compile a `cmc` assembler instruction.

**`cmova, ( mode mode -- )`** asm.sf

Compile a `cmova` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovae, ( mode mode -- )`** asm.sf

Compile a `cmovae` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovb, ( mode mode -- )`** asm.sf

Compile a `cmovb` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovbe, ( mode mode -- )`** asm.sf

Compile a `cmovbe` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovc, ( mode mode -- )`** asm.sf

Compile a `cmovc` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmove, ( mode mode -- )`** asm.sf

Compile a `cmovbe` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovg, ( mode mode -- )`** asm.sf

Compile a `cmovg` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovge, ( mode mode -- )`** asm.sf

Compile a `cmovge` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovl, ( mode mode -- )`** asm.sf

Compile a `cmovl` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovle, ( mode mode -- )`** asm.sf

Compile a `cmovle` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovna, ( mode mode -- )`** asm.sf

Compile a `cmovna` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovnae, ( mode mode -- )`** asm.sf

Compile a `cmovnae` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovnb, ( mode mode -- )`** asm.sf

Compile a `cmovnb` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovnbe, ( mode mode -- )`** asm.sf

Compile a `cmovnbe` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**`cmovnc, ( mode mode -- )`** asm.sf

Compile a `cmovnc` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.



**cmovne, ( mode mode -- )** asm.sf

Compile a `cmovne` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovng, ( mode mode -- )** asm.sf

Compile a `cmovng` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovnge, ( mode mode -- )** asm.sf

Compile a `cmovnge` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovnl, ( mode mode -- )** asm.sf

Compile a `cmovnl` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovnle, ( mode mode -- )** asm.sf

Compile a `cmovnle` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovno, ( mode mode -- )** asm.sf

Compile a `cmovno` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovnp, ( mode mode -- )** asm.sf

Compile a `cmovnp` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovns, ( mode mode -- )** asm.sf

Compile a `cmovns` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovnz, ( mode mode -- )** asm.sf

Compile a `cmovnz` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovo, ( mode mode -- )** asm.sf

Compile a `cmovo` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovp, ( mode mode -- )** asm.sf

Compile a `cmovp` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovpe, ( mode mode -- )** asm.sf

Compile a `cmovpe` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovpo, ( mode mode -- )** asm.sf

Compile a `cmovpo` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovs, ( mode mode -- )** asm.sf

Compile a `cmovs` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmovz, ( mode mode -- )** asm.sf

Compile a `cmovz` assembler instruction with the first `mode` being the destination register and the second `mode` being the source.

**cmp, ( mode mode -- )** asm.sf

Compile a `cmp` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**cmp, ( mode single -- )** asm.sf

Compile a `cmp` assembler instruction with `mode` being the destination and `single` being the immediate source operand.

**cmpsb, ( -- )** asm.sf

Compile a `cmpsb` assembler instruction.

**cmpsd, ( -- )** asm.sf

Compile a `cmpsd` assembler instruction.

**cmpsw, ( -- )** asm.sf

Compile a `cmpsw` assembler instruction.

**cmpxchg, ( mode mode -- )** asm.sf

Compile a `cmpxchg` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**`cmpxchg8b, ( mode -- )`** asm.sf

Compile a `cmpxchg8b` assembler instruction with `mode` being the destination.

**`cpuid, ( -- )`** asm.sf

Compile a `cpuid` assembler instruction.

**`cr0 ( -- creg )`** asm.sf

`creg` is control register 0.

**`cr1 ( -- creg )`** asm.sf

`creg` is control register 1.

**`cr2 ( -- creg )`** asm.sf

`creg` is control register 2.

**`cr3 ( -- creg )`** asm.sf

`creg` is control register 3.

**`cr4 ( -- creg )`** asm.sf

`creg` is control register 4.

**`creg ( stack-diagram -- 1st )`** asm.sf

When used in a stack diagram, specifies an input or output parameter with data type `creg`.

**`cs ( -- sreg )`** asm.sf

`sreg` is segment register `cs`.

**`cs: ( -- )`** asm.sf

Compile a `cs` segment override instruction prefix.

**`cwd, ( -- )`** asm.sf

Compile a `cwd` assembler instruction.

**`cwde, ( -- )`** asm.sf

Compile a `cwde` assembler instruction.

**`cx ( -- mode )`**

asm.sf

`mode` is the register direct `cx` addressing mode.

**`daa, ( -- )`**

asm.sf

Compile a `daa` assembler instruction.

**`das, ( -- )`**

asm.sf

Compile a `das` assembler instruction.

**`db, ( single -- )`**

asm.sf

Reserve space for one byte in the code space and store the least significant byte of `single` in it. An exception is thrown if the code space overflows.

**`dd, ( single -- )`**

asm.sf

Reserve space for one cell in the code space and store `single` in the cell. An exception is thrown if the code space overflows.

**`dec, ( mode -- )`**

asm.sf

Compile a `dec` assembler instruction with `mode` being the operand.

**`destination-address ( stack-diagram -- 1st )`**

asm.sf

When used in a stack diagram, specifies an input or output parameter with data type `destination-address`.

**`dh ( -- mode )`**

asm.sf

`mode` is the register direct `dh` addressing mode.

**`di ( -- mode )`**

asm.sf

`mode` is the register direct `di` addressing mode.

**`disassemble ( address -- )`**

asm.sf

Send an assembly code representation of the machine code instructions starting at `address` to the default output stream. The disassembly ends when a `ret` instruction is encountered that is not being skipped over by a conditional or unconditional jump instruction within the disassembled code.

**`div, ( mode -- )`**

asm.sf

Compile a `div` assembler instruction with `mode` being the source operand.

**`dl ( -- mode )`** asm.sf

`mode` is the register direct `dl` addressing mode.

**`dq, ( double -- )`** asm.sf

Reserve space for two cells in the code space and store `double` in the two cells. An exception is thrown if the code space overflows.

**`dr0 ( -- dreg )`** asm.sf

`dreg` is debug register 0.

**`dr1 ( -- dreg )`** asm.sf

`dreg` is debug register 1.

**`dr2 ( -- dreg )`** asm.sf

`dreg` is debug register 2.

**`dr3 ( -- dreg )`** asm.sf

`dreg` is debug register 3.

**`dr4 ( -- dreg )`** asm.sf

`dreg` is debug register 4.

**`dr5 ( -- dreg )`** asm.sf

`dreg` is debug register 5.

**`dr6 ( -- dreg )`** asm.sf

`dreg` is debug register 6.

**`dr7 ( -- dreg )`** asm.sf

`dreg` is debug register 7.

**`dreg ( stack-diagram -- 1st )`** asm.sf

When used in a stack diagram, specifies an input or output parameter with data type `dreg`.

**`ds ( -- sreg )`** asm.sf

`sreg` is segment register `ds`.

**`ds: ( -- )`**

asm.sf

Compile a `ds` segment override instruction prefix.

**`dw, ( single -- )`**

asm.sf

Reserve space for two bytes in the code space and store the lower 16 bits of `single` in the two bytes. An exception is thrown if the code space overflows.

**`dx ( -- mode )`**

asm.sf

`mode` is the register direct `dx` addressing mode.

**`ea ( mode -- unsigned )`**

asm.sf

Splits a double-cell item `mode` into two single-cell items. `unsigned` is the most significant cell of `mode`, which specifies the effective address of an addressing mode.

**`eax ( -- mode )`**

asm.sf

`mode` is the register direct `eax` addressing mode.

**`ebp ( -- mode )`**

asm.sf

`mode` is the register direct `ebp` addressing mode.

**`ebx ( -- mode )`**

asm.sf

`mode` is the register direct `ebx` addressing mode.

**`ecx ( -- mode )`**

asm.sf

`mode` is the register direct `ecx` addressing mode.

**`edi ( -- mode )`**

asm.sf

`mode` is the register direct `edi` addressing mode.

**`edx ( -- mode )`**

asm.sf

`mode` is the register direct `edx` addressing mode.

**`else, ( origin-address -- 1st )`**

asm.sf

Compile a `jmp` assembler instruction with a dummy jump destination. `1st` is a pointer to the code space location after this instruction. Resolve a forward jump by compiling the jump offset from `origin-address` to the current location of the code space.

**endcode ( code-definition -- )** asm.sf

Makes `code-definition` the definition most recently added to the current compilation vocabulary. Remove the `assembler` vocabulary from the context vocabulary list and make it the head of the hidden vocabulary list.

**enter, ( unsigned unsigned -- )** asm.sf

Compile an `enter` assembler instruction with the first `unsigned` being the size of the stack frame in bytes and the second `unsigned` being the nesting level.

**es ( -- sreg )** asm.sf

`sreg` is segment register `es`.

**es: ( -- )** asm.sf

Compile an `es` segment override instruction prefix.

**esi ( -- mode )** asm.sf

`mode` is the register direct `esi` addressing mode.

**esp ( -- mode )** asm.sf

`mode` is the register direct `esp` addressing mode.

**f2xm1, ( -- )** asm.sf

Compile a `f2xm1` assembler instruction.

**fabs, ( -- )** asm.sf

Compile a `fabs` assembler instruction.

**fadd, ( mode -- )** asm.sf

Compile a `fadd` assembler instruction with `st0` being the destination and `mode` being the source floating point register.

**fadd, ( mode mode -- )** asm.sf

Compile a `fadd` assembler instruction with the first `mode` being the destination floating-point register and the second `mode` being the source floating-point register.

**faddp, ( -- )** asm.sf

Compile a `faddp` assembler instruction with `st1` being the destination and `st0` being the source.

**faddp, ( mode mode -- )**

Compile a `faddp` assembler instruction with the first `mode` being the destination floating-point register and the second `mode` being the source floating-point register.

**fbld, ( mode -- )**

asm.sf

Compile a `fbld` assembler instruction with `mode` being the source.

**fbstp, ( mode -- )**

asm.sf

Compile a `fbstp` assembler instruction with `mode` being the destination.

**fchs, ( -- )**

asm.sf

Compile a `fchs` assembler instruction.

**fclex, ( -- )**

asm.sf

Compile a `fclex` assembler instruction.

**fcmovb, ( mode mode -- )**

asm.sf

Compile a `fcmovb` assembler instruction with the first `mode` being the source and the second `mode` being the destination.

**fcmovbe, ( mode mode -- )**

asm.sf

Compile a `fcmovbe` assembler instruction with the first `mode` being the source and the second `mode` being the destination.

**fcmove, ( mode mode -- )**

asm.sf

Compile a `fcmove` assembler instruction with the first `mode` being the source and the second `mode` being the destination.

**fcmovnb, ( mode mode -- )**

asm.sf

Compile a `fcmovnb` assembler instruction with the first `mode` being the source and the second `mode` being the destination.

**fcmovnbe, ( mode mode -- )**

asm.sf

Compile a `fcmovnbe` assembler instruction with the first `mode` being the source and the second `mode` being the destination.

**fcmovne, ( mode mode -- )**

asm.sf



Compile a `fcmovne` assembler instruction with the first mode being the source and the second mode being the destination.

**fcmovnu, ( mode mode -- )**

asm.sf

Compile a `fcmovnu` assembler instruction with the first mode being the source and the second mode being the destination.

**fcmovu, ( mode mode -- )**

asm.sf

Compile a `fcmovu` assembler instruction with the first mode being the source and the second mode being the destination.

**fcom, ( -- )**

asm.sf

Compile a `fcom` assembler instruction with `st1` being the destination and `st0` being the source.

**fcom, ( mode -- )**

asm.sf

Compile a `fcom` assembler instruction with `mode` being the destination and `st0` being the source.

**fcom, ( mode mode -- )**

asm.sf

Compile a `fcom` assembler instruction with the first mode being the destination and the second mode being the source.

**fcomi, ( mode mode -- )**

asm.sf

Compile a `fcomi` assembler instruction with the first mode being the destination and the second mode being the source.

**fcomip, ( mode mode -- )**

asm.sf

Compile a `fcomip` assembler instruction with the first mode being the destination and the second mode being the source.

**fcomp, ( -- )**

asm.sf

Compile a `fcomp` assembler instruction with `st1` being the destination and `st0` being the source.

**fcomp, ( mode -- )**

asm.sf

Compile a `fcomp` assembler instruction with `mode` as the destination and `st0` being the source.

**fcomp, ( mode mode -- )**

asm.sf

Compile a `fcomp` assembler instruction with the first mode being the destination and the second mode being the source.

<b>fcompp, ( -- )</b>	asm.sf
Compile a <code>fcompp</code> assembler instruction with <code>st1</code> being the destination and <code>st0</code> being the source.	
<b>fcos, ( -- )</b>	asm.sf
Compile a <code>fcos</code> assembler instruction.	
<b>fdecstp, ( -- )</b>	asm.sf
Compile a <code>fdecstp</code> assembler instruction.	
<b>fdiv, ( mode -- )</b>	asm.sf
Compile a <code>fdiv</code> assembler instruction with <code>mode</code> being the destination and <code>st0</code> being the source.	
<b>fdiv, ( mode mode -- )</b>	asm.sf
Compile a <code>fdiv</code> assembler instruction with the first <code>mode</code> being the destination floating-point register and the second <code>mode</code> being the source floating-point register.	
<b>fdivp, ( -- )</b>	asm.sf
Compile a <code>fdivp</code> assembler instruction with <code>st1</code> being the destination and <code>st0</code> being the source.	
<b>fdivp, ( mode mode -- )</b>	asm.sf
Compile a <code>fdivp</code> assembler instruction with the first <code>mode</code> being the destination floating-point register and the second <code>mode</code> being the source floating-point register.	
<b>fdivr, ( mode -- )</b>	asm.sf
Compile a <code>fdivr</code> assembler instruction with <code>mode</code> being the destination floating point register and <code>st0</code> being the source.	
<b>fdivr, ( mode mode -- )</b>	asm.sf
Compile a <code>fdivr</code> assembler instruction with the first <code>mode</code> being the destination floating-point register and the second <code>mode</code> being the source floating-point register.	
<b>fdivrp, ( -- )</b>	asm.sf
Compile a <code>fdivrp</code> assembler instruction with <code>st1</code> being the destination and <code>st0</code> being the source.	
<b>fdivrp, ( mode mode -- )</b>	asm.sf
Compile a <code>fdivrp</code> assembler instruction with the first <code>mode</code> being the destination floating-point register and the second <code>mode</code> being the source floating-point register.	

<b>ffree, ( mode -- )</b>	asm.sf
Compile a <code>ffree</code> assembler instruction with <code>mode</code> being the affected floating-point register.	
<b>fiadd, ( mode -- )</b>	asm.sf
Compile a <code>fiadd</code> assembler instruction with <code>st0</code> being the destination and <code>mode</code> being the source.	
<b>ficom, ( mode -- )</b>	asm.sf
Compile a <code>ficom</code> assembler instruction with <code>st0</code> being the destination and <code>mode</code> being the source.	
<b>ficomp, ( mode -- )</b>	asm.sf
Compile a <code>ficomp</code> assembler instruction with <code>st0</code> being the destination and <code>mode</code> being the source.	
<b>fidiv, ( mode -- )</b>	asm.sf
Compile a <code>fidiv</code> assembler instruction with <code>st0</code> being the destination and <code>mode</code> being the source.	
<b>fidivr, ( mode -- )</b>	asm.sf
Compile a <code>fidivr</code> assembler instruction with <code>st0</code> being the destination and <code>mode</code> being the source.	
<b>field, ( mode -- )</b>	asm.sf
Compile a <code>field</code> assembler instruction with <code>mode</code> being the source.	
<b>fimul, ( mode -- )</b>	asm.sf
Compile a <code>fiadd</code> assembler instruction with <code>st0</code> being the destination and <code>mode</code> being the source.	
<b>fincstp, ( -- )</b>	asm.sf
Compile a <code>fincstp</code> assembler instruction.	
<b>finit, ( -- )</b>	asm.sf
Compile a <code>finit</code> assembler instruction.	
<b>fist, ( mode -- )</b>	asm.sf
Compile a <code>fist</code> assembler instruction with <code>mode</code> being the destination.	

**fistp, ( mode -- )** asm.sf

Compile a `fistp` assembler instruction with `mode` being the destination.

**fisttp, ( mode -- )** asm.sf

Compile a `fisttp` assembler instruction with `mode` being the destination.

**fisub, ( mode -- )** asm.sf

Compile a `fisub` assembler instruction with `st0` being the destination and `mode` being the source.

**fisubr, ( mode -- )** asm.sf

Compile a `fisubr` assembler instruction with `st0` being the destination and `mode` being the source.

**fld, ( mode -- )** asm.sf

Compile a `fld` assembler instruction with `mode` being the source.

**fld1, ( -- )** asm.sf

Compile a `fld1` assembler instruction.

**fldcw, ( mode -- )** asm.sf

Compile a `fldcw` assembler instruction with `mode` being the source.

**fldenv, ( mode -- )** asm.sf

Compile a `fldenv` assembler instruction with `mode` being the source.

**fldenvw, ( mode -- )**

Compile a `fldenv` assembler instruction with `mode` being the 16-bit source.

**fldl2e, ( -- )** asm.sf

Compile a `fldl2e` assembler instruction.

**fldl2t, ( -- )** asm.sf

Compile a `fldl2t` assembler instruction.

**fldlg2, ( -- )** asm.sf

Compile a `fldlg2` assembler instruction.

<b>fldln2, ( -- )</b>	asm.sf
Compile a <code>fldln2</code> assembler instruction.	
<b>fldpi, ( -- )</b>	asm.sf
Compile a <code>fldpi</code> assembler instruction.	
<b>fldz, ( -- )</b>	asm.sf
Compile a <code>fldz</code> assembler instruction.	
<b>float: ( stack-diagram -- 1st )</b>	asm.sf
When used in a stack diagram, specifies the succeeding input or output parameter shall be assigned to the hardware floating-point stack. An exception is thrown if the input or output parameter is not a floating-point number.	
<b>fmul, ( mode -- )</b>	asm.sf
Compile a <code>fmul</code> assembler instruction with <code>mode</code> being the destination and <code>st0</code> being the source.	
<b>fmul, ( mode mode -- )</b>	asm.sf
Compile a <code>fmul</code> assembler instruction with the first <code>mode</code> being the destination floating-point register and the second <code>mode</code> being the source floating-point register.	
<b>fmulp, ( -- )</b>	asm.sf
Compile a <code>fmulp</code> assembler instruction with <code>st1</code> being the destination and <code>st0</code> being the source.	
<b>fmulp, ( mode mode -- )</b>	asm.sf
Compile a <code>fmulp</code> assembler instruction with the first <code>mode</code> being the destination floating-point register and the second <code>mode</code> being the source floating-point register.	
<b>fnclx, ( -- )</b>	asm.sf
Compile a <code>fnclx</code> assembler instruction.	
<b>fninit, ( -- )</b>	asm.sf
Compile a <code>fninit</code> assembler instruction.	
<b>fnop, ( -- )</b>	asm.sf
Compile a <code>fnop</code> assembler instruction.	

<b>fnsave, ( mode -- )</b>	asm.sf
Compile a <code>fnsave</code> assembler instruction with <code>mode</code> being the destination.	
<b>fnsavew, ( mode -- )</b>	asm.sf
Compile a <code>fnsave</code> assembler instruction with <code>mode</code> being the 16-bit destination.	
<b>fnstcw, ( mode -- )</b>	asm.sf
Compile a <code>fnstcw</code> assembler instruction with <code>mode</code> being the destination.	
<b>fnstenv, ( mode -- )</b>	asm.sf
Compile a <code>fnstenv</code> assembler instruction with <code>mode</code> being the destination.	
<b>fnstenvw, ( mode -- )</b>	asm.sf
Compile a <code>fnstenv</code> assembler instruction with <code>mode</code> being the 16-bit destination.	
<b>fnstsw, ( mode -- )</b>	asm.sf
Compile a <code>fnstsw</code> assembler instruction with <code>mode</code> being the 16-bit destination.	
<b>fpatan, ( -- )</b>	asm.sf
Compile a <code>fpatan</code> assembler instruction.	
<b>fprem, ( -- )</b>	asm.sf
Compile a <code>fprem</code> assembler instruction.	
<b>fprem1, ( -- )</b>	asm.sf
Compile a <code>fprem1</code> assembler instruction.	
<b>fptan, ( -- )</b>	asm.sf
Compile a <code>fptan</code> assembler instruction.	
<b>frndint, ( -- )</b>	asm.sf
Compile a <code>frndint</code> assembler instruction.	
<b>frstor, ( mode -- )</b>	asm.sf
Compile a <code>frstor</code> assembler instruction with <code>mode</code> being the source.	
<b>frstorw, ( mode -- )</b>	asm.sf

Compile a `fstor` assembler instruction with `mode` being the 16-bit source.

**`fs ( -- sreg )`** asm.sf

`sreg` is segment register `fs`.

**`fs: ( -- )`** asm.sf

Compile a `fs` segment override instruction prefix.

**`fsave, ( mode -- )`** asm.sf

Compile a `fsave` assembler instruction with `mode` being the destination.

**`fscale, ( -- )`** asm.sf

Compile a `fscale` assembler instruction.

**`fsin, ( -- )`** asm.sf

Compile a `fsin` assembler instruction.

**`fsincos, ( -- )`** asm.sf

Compile a `fsincos` assembler instruction.

**`fsqrt, ( -- )`** asm.sf

Compile a `fsqrt` assembler instruction.

**`fst, ( mode -- )`** asm.sf

Compile a `fst` assembler instruction with `mode` being the destination.

**`fstcw, ( mode -- )`** asm.sf

Compile a `fstcw` assembler instruction with `mode` being the destination.

**`fstenv, ( mode -- )`** asm.sf

Compile a `fstenv` assembler instruction with `mode` being the destination.

**`fstenvw, ( mode -- )`** asm.sf

Compile a `fstenvw` assembler instruction with `mode` being the 16-bit destination.

**`fstp, ( mode -- )`** asm.sf

Compile a `fstp` assembler instruction with `mode` being the destination.

**fstsw, ( mode -- )** asm.sf

Compile a `fstsw` assembler instruction with `mode` being the 16-bit destination.

**fsub, ( mode -- )** asm.sf

Compile a `fsub` assembler instruction with `st0` being the destination and `mode` being the source floating point register.

**fsub, ( mode mode -- )** asm.sf

Compile a `fsub` assembler instruction with the first `mode` being the destination floating-point register and the second `mode` being the source floating-point register.

**fsubp, ( -- )** asm.sf

Compile a `fsubp` assembler instruction with `st1` being the destination and `st0` being the source.

**fsubp, ( mode mode -- )** asm.sf

Compile a `fsubp` assembler instruction with the first `mode` being the destination floating-point register and the second `mode` being the source floating-point register.

**fsubr, ( mode -- )** asm.sf

Compile a `fsubr` assembler instruction with `mode` being the destination floating point register and `st0` being the source.

**fsubr, ( mode mode -- )** asm.sf

Compile a `fsubr` assembler instruction with the first `mode` being the destination floating-point register and the second `mode` being the source floating-point register.

**fsubrp, ( -- )** asm.sf

Compile a `fsubrp` assembler instruction with `st1` being the destination and `st0` being the source.

**fsubrp, ( mode mode -- )** asm.sf

Compile a `fsubrp` assembler instruction with the first `mode` being the destination floating-point register and the second `mode` being the source floating-point register.

**ftst, ( -- )** asm.sf

Compile a `ftst` assembler instruction.

**fucom, ( -- )** asm.sf



Compile a `fucom` assembler instruction with `st1` being the destination and `st0` being the source.

**`fucom, ( mode -- )`**

asm.sf

Compile a `fucom` assembler instruction with `mode` being the destination and `st0` being the source.

**`fucomi, ( mode mode -- )`**

asm.sf

Compile a `fucomi` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**`fucomip, ( mode mode -- )`**

asm.sf

Compile a `fucomip` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**`fucomp, ( -- )`**

asm.sf

Compile a `fucomp` assembler instruction with `st1` being the destination and `st0` being the source.

**`fucomp, ( mode -- )`**

asm.sf

Compile a `fucomp` assembler instruction with `mode` being the destination and `st0` being the source.

**`fucompp, ( -- )`**

asm.sf

Compile a `fucompp` assembler instruction with `st1` being the destination and `st0` being the source.

**`fwait, ( -- )`**

asm.sf

Compile a `fwait` assembler instruction.

**`fxam, ( -- )`**

asm.sf

Compile a `fxam` assembler instruction.

**`fxch, ( -- )`**

asm.sf

Compile a `fxch` assembler instruction with `st0` and `st1` being the two floating-point registers.

**`fxch, ( mode -- )`**

asm.sf

Compile a `fxch` assembler instruction with `mode` and `st0` being the two floating-point registers.

**`fxch, ( mode mode -- )`**

asm.sf

Compile a `fxch` assembler instruction with the first and the second `mode` being the two floating-point registers.

**`fxtract, ( -- )`**

asm.sf

Compile a `fxtract` assembler instruction.

**`fy12x, ( -- )`**

asm.sf

Compile a `fy12x` assembler instruction.

**`fy12xp1, ( -- )`**

asm.sf

Compile a `fy12xp1` assembler instruction.

**`getsec, ( -- )`**

asm.sf

Compile a `getsec` assembler instruction.

**`gs ( -- sreg )`**

asm.sf

`sreg` is segment register `gs`.

**`gs: ( -- )`**

asm.sf

Compile a `gs` segment override instruction prefix.

**`hlt, ( -- )`**

asm.sf

Compile a `hlt` assembler instruction.

**`idiv, ( mode -- )`**

asm.sf

Compile an `idiv` assembler instruction with `mode` being the source operand.

**`ifa, ( -- origin-address )`**

asm.sf

Compile a `jna` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**`ifae, ( -- origin-address )`**

asm.sf

Compile a `jnae` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**`ifb, ( -- origin-address )`**

asm.sf

Compile a `jnb` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifbe, ( -- origin-address )** asm.sf

Compile a `jnb` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifc, ( -- origin-address )** asm.sf

Compile a `jnc` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ife, ( -- origin-address )** asm.sf

Compile a `jne` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifg, ( -- origin-address )** asm.sf

Compile a `jng` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifge, ( -- origin-address )** asm.sf

Compile a `jnge` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifl, ( -- origin-address )** asm.sf

Compile a `jnl` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifle, ( -- origin-address )** asm.sf

Compile a `jnle` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifna, ( -- origin-address )** asm.sf

Compile a `ja` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnae, ( -- origin-address )** asm.sf

Compile a `jae` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnb, ( -- origin-address )** asm.sf

Compile a `jnb` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnbe, ( -- origin-address )** asm.sf

Compile a `jbe` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnc, ( -- origin-address )** asm.sf

Compile a `jc` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifncxz, ( -- origin-address )** asm.sf

Compile a `jcxz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifne, ( -- origin-address )** asm.sf

Compile a `je` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnecxz, ( -- origin-address )** asm.sf

Compile a `jecxz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifng, ( -- origin-address )** asm.sf

Compile a `jg` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnge, ( -- origin-address )** asm.sf

Compile a `jge` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnl, ( -- origin-address )** asm.sf

Compile a `jnl` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnle, ( -- origin-address )** asm.sf

Compile a `jle` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifno, ( -- origin-address )** asm.sf

Compile a `jno` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnp, ( -- origin-address )** asm.sf

Compile a `jmp` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifns, ( -- origin-address )** asm.sf

Compile a `js` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifnz, ( -- origin-address )** asm.sf

Compile a `jz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifo, ( -- origin-address )** asm.sf

Compile a `jno` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifp, ( -- origin-address )** asm.sf

Compile a `jnp` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifpe, ( -- origin-address )** asm.sf

Compile a `jpo` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifpo, ( -- origin-address )** asm.sf

Compile a `jpe` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifs, ( -- origin-address )** asm.sf

Compile a `jns` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**ifz, ( -- origin-address )** asm.sf

Compile a `jnz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction.

**imul, ( mode -- )** asm.sf

Compile an `imul` assembler instruction with `mode` being the source operand.

**imul, ( mode mode -- )** asm.sf

Compile an `imul` assembler instruction with the first `mode` being the destination operand and the second `mode` being the source operand.

**imul, ( mode mode single -- )** asm.sf

Compile an `imul` assembler instruction with the first `mode` being the destination operand, the second `mode` being the first source operand and `single` being the second (immediate) source operand.

**in, ( mode mode -- )** asm.sf

Compile an `in` assembler instruction with the first `mode` specifying the destination register and the second `mode` specifying register `dx` as the port number. An exception is thrown if the second `mode` is not register `dx`.

**in, ( mode unsigned -- )** asm.sf

Compile an `in` assembler instruction with `mode` specifying the destination register and `unsigned` specifying the immediate port number.

**inc, ( mode -- )** asm.sf

Compile an `inc` assembler instruction with `mode` being the operand.

**index[eax\*2] ( mode -- mode )** asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 2 times `[eax]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**index[eax\*4] ( mode -- mode )** asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 4 times `[eax]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**index[eax\*8] ( mode -- mode )** asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 8 times `[eax]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**index[eax] ( mode -- mode )** asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and `[eax]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**index[ebp\*2] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 2 times [ebp] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebp\*4] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 4 times [ebp] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebp\*8] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 8 times [ebp] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebp] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and [ebp] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebx\*2] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 2 times [ebx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebx\*4] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 4 times [ebx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebx\*8] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 8 times [ebx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ebx] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and [ebx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ecx\*2] ( mode -- mode )** asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 2 times [ecx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ecx\*4] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 4 times [ecx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ecx\*8] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 8 times [ecx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[ecx] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and [ecx] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[edi\*2] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 2 times [edi] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[edi\*4] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 4 times [edi] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[edi\*8] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and 8 times [edi] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[edi] ( mode -- mode )**

asm.sf

mode is the base and scaled index addressing mode with the input parameter mode being the base and [edi] being the index. An exception is thrown if the input parameter mode is not a register indirect or a register indirect with displacement addressing mode.

**index[edx\*2] ( mode -- mode )**

asm.sf



`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 2 times `[edx]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[edx*4] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 4 times `[edx]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[edx*8] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 8 times `[edx]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[edx] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and `[edx]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[esi*2] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 2 times `[esi]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[esi*4] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 4 times `[esi]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[esi*8] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and 8 times `[esi]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`index[esi] ( mode -- mode )`**

asm.sf

`mode` is the base and scaled index addressing mode with the input parameter `mode` being the base and `[esi]` being the index. An exception is thrown if the input parameter `mode` is not a register indirect or a register indirect with displacement addressing mode.

**`insb, ( -- )`**

asm.sf

Compile an `insb` assembler instruction.

<b>insd, ( -- )</b>	asm.sf
Compile an <code>insd</code> assembler instruction.	
<b>insw, ( -- )</b>	asm.sf
Compile an <code>insw</code> assembler instruction.	
<b>int, ( unsigned -- )</b>	asm.sf
Compile an <code>int</code> assembler instruction with <code>unsigned</code> being the interrupt vector.	
<b>into, ( -- )</b>	asm.sf
Compile a <code>into</code> assembler instruction.	
<b>invd, ( -- )</b>	asm.sf
Compile an <code>invd</code> assembler instruction.	
<b>invlpg, ( mode -- )</b>	asm.sf
Compile an <code>invlpg</code> assembler instruction with <code>mode</code> being the source operand.	
<b>iret, ( -- )</b>	asm.sf
Compile an <code>iret</code> assembler instruction.	
<b>iretd, ( -- )</b>	asm.sf
Compile an <code>iretd</code> assembler instruction.	
<b>ja, ( address -- )</b>	asm.sf
Compile a <code>ja</code> assembler instruction with <code>address</code> being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.	
<b>jae, ( address -- )</b>	asm.sf
Compile a <code>jae</code> assembler instruction with <code>address</code> being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.	
<b>jb, ( address -- )</b>	asm.sf
Compile a <code>jb</code> assembler instruction with <code>address</code> being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.	

**jbe, ( address -- )**

asm.sf

Compile a `jbe` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jc, ( address -- )**

asm.sf

Compile a `jc` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jcxz, ( address -- )**

asm.sf

Compile a `jcxz` assembler instruction with `address` being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.

**je, ( address -- )**

asm.sf

Compile a `je` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jecxz, ( address -- )**

asm.sf

Compile a `jecxz` assembler instruction with `address` being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.

**jg, ( address -- )**

asm.sf

Compile a `jg` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jge, ( address -- )**

asm.sf

Compile a `jge` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**j1, ( address -- )**

asm.sf

Compile a `j1` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jle, ( address -- )**

asm.sf

Compile a `jle` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jmp, ( address -- )** asm.sf

Compile an unconditional `jmp` instruction to destination `address`. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jmp, ( mode -- )** asm.sf

Compile a `jmp` instruction to destination `mode`.

**jmpf, ( mode -- )** asm.sf

Compile a far `jmp` instruction to destination `mode`.

**jmpf, ( segment address -- )** asm.sf

Compile a far `jmp` instruction to the destination specified by `segment` and `address`.

**jna, ( address -- )** asm.sf

Compile a `jna` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jnae, ( address -- )** asm.sf

Compile a `jnae` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jnb, ( address -- )** asm.sf

Compile a `jnb` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jnb, ( address -- )** asm.sf

Compile a `jnb` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jnc, ( address -- )** asm.sf

Compile a `jnc` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jne, ( address -- )** asm.sf

Compile a `jne` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jng, ( address -- )`**

asm.sf

Compile a `jng` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jnge, ( address -- )`**

asm.sf

Compile a `jnge` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jnl, ( address -- )`**

asm.sf

Compile a `jnl` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jnle, ( address -- )`**

asm.sf

Compile a `jnle` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jno, ( address -- )`**

asm.sf

Compile a `jno` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jnp, ( address -- )`**

asm.sf

Compile a `jnp` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jns, ( address -- )`**

asm.sf

Compile a `jns` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**`jnz, ( address -- )`**

asm.sf

Compile a `jnz` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jo, ( address -- )** asm.sf

Compile a `jo` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jp, ( address -- )** asm.sf

Compile a `jp` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jpe, ( address -- )** asm.sf

Compile a `jpe` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jpo, ( address -- )** asm.sf

Compile a `jpo` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**js, ( address -- )** asm.sf

Compile a `js` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**jz, ( address -- )** asm.sf

Compile a `jz` assembler instruction with `address` being the jump destination. If the destination address is within the range of a relative short jump, compile a relative short jump with a byte displacement. Otherwise, compile a relative near jump with a 32-bit displacement.

**lahf, ( -- )** asm.sf

Compile a `lahf` assembler instruction.

**lar, ( mode mode -- )** asm.sf

Compile a `lar` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**lds, ( mode mode -- )** asm.sf

Compile a `lds` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**lea, ( mode mode -- )** asm.sf

Compile a `lea` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**leave, ( -- )** asm.sf

Compile a `leave` assembler instruction.

**les, ( mode mode -- )** asm.sf

Compile a `les` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**lfs, ( mode mode -- )** asm.sf

Compile a `lfs` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**lgdt, ( mode -- )** asm.sf

Compile a `lgdt` assembler instruction with `mode` being the source operand.

**lgs, ( mode mode -- )** asm.sf

Compile a `lgs` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**lidt, ( mode -- )** asm.sf

Compile a `lidt` assembler instruction with `mode` being the source operand.

**lldt, ( mode -- )** asm.sf

Compile a `lldt` assembler instruction with `mode` being the source operand.

**lmsw, ( mode -- )** asm.sf

Compile a `lmsw` assembler instruction with `mode` being the source operand.

**location ( -- address -> address )** asm.sf

`address -> address` is the address of a cell containing the disassembler's location counter.

**lock ( -- )** asm.sf

Compile a `lock` assembler instruction prefix.

<b>lods b, ( -- )</b>	asm.sf
Compile a <code>lods b</code> assembler instruction.	
<b>lods d, ( -- )</b>	asm.sf
Compile a <code>lods d</code> assembler instruction.	
<b>lods w, ( -- )</b>	asm.sf
Compile a <code>lods w</code> assembler instruction.	
<b>loop, ( address -- )</b>	asm.sf
Compile a <code>loop</code> assembler instruction with <code>address</code> being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.	
<b>loope, ( address -- )</b>	asm.sf
Compile a <code>loope</code> assembler instruction with <code>address</code> being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.	
<b>loopne, ( address -- )</b>	asm.sf
Compile a <code>loopne</code> assembler instruction with <code>address</code> being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.	
<b>loopnz, ( address -- )</b>	asm.sf
Compile a <code>loopnz</code> assembler instruction with <code>address</code> being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.	
<b>loopz, ( address -- )</b>	asm.sf
Compile a <code>loopz</code> assembler instruction with <code>address</code> being the jump destination. An exception is thrown if the destination address is not within the range of a relative short jump.	
<b>lsl, ( mode mode -- )</b>	asm.sf
Compile a <code>lsl</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being the source.	
<b>lss, ( mode mode -- )</b>	asm.sf
Compile a <code>lss</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being the source.	
<b>ltr, ( mode -- )</b>	asm.sf
Compile a <code>ltr</code> assembler instruction with <code>mode</code> being the source operand.	



**lzcnt, ( mode mode -- )** asm.sf

Compile a `lzcnt` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**m16 ( mode -- 1st )** asm.sf

Specifies that `mode` refers to a 16-bit memory operand, returning `1st`. An exception is thrown if `mode` is a general-purpose register, a floating-point register or a byte operand.

**m32 ( mode -- 1st )** asm.sf

Specifies that `mode` refers to a 32-bit memory operand, returning `1st`. An exception is thrown if `mode` is a general-purpose register, a floating-point register or a byte operand.

**m64 ( mode -- 1st )** asm.sf

Specifies that `mode` refers to a 64-bit memory operand, returning `1st`. An exception is thrown if `mode` is a general-purpose register, a floating-point register or a byte operand.

**m80 ( mode -- 1st )** asm.sf

Specifies that `mode` refers to a 80-bit memory operand, returning `1st`. An exception is thrown if `mode` is a general-purpose register, a floating-point register or a byte operand.

**mode ( stack-diagram -- 1st )** asm.sf

When used in a stack diagram, specifies an input or output parameter with data type `mode`.

**mov, ( creg mode -- )** asm.sf

Compile a `mov` assembler instruction with `creg` being the destination control register and `mode` being the source register.

**mov, ( dreg mode -- )** asm.sf

Compile a `mov` assembler instruction with `dreg` being the destination debug register and `mode` being the source register.

**mov, ( mode creg -- )** asm.sf

Compile a `mov` assembler instruction with `mode` being the destination register and `creg` being the source control register.

**mov, ( mode dreg -- )** asm.sf

Compile a `mov` assembler instruction with `mode` being the destination register and `dreg` being the source debug register.

**mov, ( mode mode -- )** asm.sf

Compile a `mov` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**mov, ( mode single -- )** asm.sf

Compile a `mov` assembler instruction with `mode` being the destination and `single` being the immediate source operand.

**mov, ( mode sreg -- )** asm.sf

Compile a `mov` assembler instruction with `mode` being the destination and `sreg` being the source segment register.

**mov, ( sreg mode -- )** asm.sf

Compile a `mov` assembler instruction with `sreg` being the destination segment register and `mode` being the source.

**movsb, ( -- )** asm.sf

Compile a `movsb` assembler instruction.

**movsd, ( -- )** asm.sf

Compile a `movsd` assembler instruction.

**movsw, ( -- )** asm.sf

Compile a `movsw` assembler instruction.

**movsx, ( mode mode -- )** asm.sf

Compile a `movsx` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**movzx, ( mode mode -- )** asm.sf

Compile a `movsx` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

**mul, ( mode -- )** asm.sf

Compile a `mul` assembler instruction with `mode` being the source operand.

**neg, ( mode -- )** asm.sf

Compile a `neg` assembler instruction with `mode` being the operand.

<b>nop, ( -- )</b>	asm.sf
Compile a <code>nop</code> assembler instruction.	
<b>nop, ( mode -- )</b>	asm.sf
Compile a <code>nop</code> assembler instruction with <code>mode</code> being a dummy operand.	
<b>not, ( mode -- )</b>	asm.sf
Compile a <code>not</code> assembler instruction with <code>mode</code> being the operand.	
<b>op ( mode -- unsigned )</b>	asm.sf
Splits a double-cell item <code>mode</code> into two single-cell items. <code>unsigned</code> is the least significant cell of <code>mode</code> , which specifies the operand of an addressing mode.	
<b>or, ( mode mode -- )</b>	asm.sf
Compile an <code>or</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being the source.	
<b>or, ( mode single -- )</b>	asm.sf
Compile an <code>or</code> assembler instruction with <code>mode</code> being the destination and <code>single</code> being the immediate source.	
<b>origin-address ( stack-diagram -- 1st )</b>	asm.sf
When used in a stack diagram, specifies an input or output parameter with data type <code>origin-address</code> .	
<b>out, ( mode mode -- )</b>	asm.sf
Compile an <code>out</code> assembler instruction with the first <code>mode</code> specifying register <code>dx</code> as the port number and the second <code>mode</code> specifying the source register. An exception is thrown if the second <code>mode</code> is not register <code>dx</code> .	
<b>out, ( unsigned mode -- )</b>	asm.sf
Compile an <code>out</code> assembler instruction with <code>unsigned</code> specifying the immediate port number and <code>mode</code> specifying the source register.	
<b>outsb, ( -- )</b>	asm.sf
Compile an <code>outsb</code> assembler instruction.	
<b>outsd, ( -- )</b>	asm.sf
Compile an <code>outsd</code> assembler instruction.	

<b>outsw, ( -- )</b>	asm.sf
Compile an outsw assembler instruction.	
<b>pop, ( mode -- )</b>	asm.sf
Compile a pop assembler instruction with mode being the destination.	
<b>pop, ( sreg -- )</b>	asm.sf
Compile a pop assembler instruction with sreg being the destination segment register.	
<b>popa, ( -- )</b>	asm.sf
Compile a popa assembler instruction.	
<b>popad, ( -- )</b>	asm.sf
Compile a popad assembler instruction.	
<b>popf, ( -- )</b>	asm.sf
Compile a popf assembler instruction.	
<b>popfd, ( -- )</b>	asm.sf
Compile a popfd assembler instruction.	
<b>push, ( mode -- )</b>	asm.sf
Compile a push assembler instruction with mode specifying the source.	
<b>push, ( single -- )</b>	asm.sf
Compile a push assembler instruction with single specifying the immediate source operand.	
<b>push, ( sreg -- )</b>	asm.sf
Compile a push assembler instruction with sreg specifying the segment register.	
<b>pusha, ( -- )</b>	asm.sf
Compile a pusha assembler instruction.	
<b>pushad, ( -- )</b>	asm.sf
Compile a pushad assembler instruction.	

<b>pushf, ( -- )</b>	asm.sf
Compile a <code>pushf</code> assembler instruction.	
<b>pushfd, ( -- )</b>	asm.sf
Compile a <code>pushfd</code> assembler instruction.	
<b>rcl, ( mode mode -- )</b>	asm.sf
Compile a <code>rcl</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being register <code>cl</code> as the count operand. An exception is thrown if the second <code>mode</code> is not register <code>cl</code> .	
<b>rcl, ( mode unsigned -- )</b>	asm.sf
Compile a <code>rcl</code> assembler instruction with <code>mode</code> being the destination and <code>unsigned</code> being the immediate count operand.	
<b>rcr, ( mode mode -- )</b>	asm.sf
Compile a <code>rcr</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being register <code>cl</code> as the count operand. An exception is thrown if the second <code>mode</code> is not register <code>cl</code> .	
<b>rcr, ( mode unsigned -- )</b>	asm.sf
Compile a <code>rcr</code> assembler instruction with <code>mode</code> being the destination and <code>unsigned</code> being the immediate count operand.	
<b>rdmsr, ( -- )</b>	asm.sf
Compile a <code>rdmsr</code> assembler instruction.	
<b>rdpmc, ( -- )</b>	asm.sf
Compile a <code>rdpmc</code> assembler instruction.	
<b>rdtsc, ( -- )</b>	asm.sf
Compile a <code>rdtsc</code> assembler instruction.	
<b>registers! ( logical stack-diagram -- )</b>	
Force the register or registers specified by <code>logical</code> onto the next data type added to <code>stack-diagram</code> .	
<b>rep ( -- )</b>	asm.sf
Compile a <code>rep</code> assembler instruction prefix.	

<b>repe ( -- )</b>	asm.sf
Compile a <code>repe</code> assembler instruction prefix.	
<b>repeat, ( origin-address destination-address -- )</b>	asm.sf
Compile a <code>jmp</code> assembler instruction with <code>destination-address</code> being the jump destination. Resolve a forward jump by compiling the jump offset from <code>origin-address</code> to the current position of the code space.	
<b>repne ( -- )</b>	asm.sf
Compile a <code>repne</code> assembler instruction prefix.	
<b>repnz ( -- )</b>	asm.sf
Compile a <code>repnz</code> assembler instruction prefix.	
<b>repz ( -- )</b>	asm.sf
Compile a <code>repz</code> assembler instruction prefix.	
<b>ret, ( -- )</b>	asm.sf
Compile a <code>ret</code> assembler instruction.	
<b>ret, ( unsigned -- )</b>	asm.sf
Compile a <code>ret</code> assembler instruction with <code>unsigned</code> being the number of stack bytes to be released.	
<b>retf, ( -- )</b>	asm.sf
Compile a far <code>ret</code> assembler instruction.	
<b>retf, ( unsigned -- )</b>	asm.sf
Compile a far <code>ret</code> assembler instruction with <code>unsigned</code> being the number of stack bytes to be released.	
<b>rol, ( mode mode -- )</b>	asm.sf
Compile a <code>rol</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being register <code>cl</code> as the count operand. An exception is thrown if the second <code>mode</code> is not register <code>cl</code> .	
<b>rol, ( mode unsigned -- )</b>	asm.sf

Compile a `rol` assembler instruction with `mode` being the destination and `unsigned` being the immediate count operand.

**`ror, ( mode mode -- )`**

asm.sf

Compile a `ror` assembler instruction with the first `mode` being the destination and the second `mode` being register `c1` as the count operand. An exception is thrown if the second `mode` is not register `c1`.

**`ror, ( mode unsigned -- )`**

asm.sf

Compile a `ror` assembler instruction with `mode` being the destination and `unsigned` being the immediate count operand.

**`rsm, ( -- )`**

asm.sf

Compile a `rsm` assembler instruction.

**`sahf, ( -- )`**

asm.sf

Compile a `sahf` assembler instruction.

**`sal, ( mode mode -- )`**

asm.sf

Compile a `sal` assembler instruction with the first `mode` being the destination and the second `mode` being register `c1` as the count operand. An exception is thrown if the second `mode` is not register `c1`.

**`sal, ( mode unsigned -- )`**

asm.sf

Compile a `sal` assembler instruction with `mode` being the destination and `unsigned` being the immediate count operand.

**`sar, ( mode mode -- )`**

asm.sf

Compile a `sar` assembler instruction with the first `mode` being the destination and the second `mode` being register `c1` as the count operand. An exception is thrown if the second `mode` is not register `c1`.

**`sar, ( mode unsigned -- )`**

asm.sf

Compile a `sar` assembler instruction with `mode` being the destination and `unsigned` being the immediate count operand.

**`sbb, ( mode mode -- )`**

asm.sf

Compile a `sbb` assembler instruction with the first `mode` being the destination and the second `mode` being the source.

<b>sbb, ( mode single -- )</b>	asm.sf
Compile a sbb assembler instruction with mode being the destination and single being the immediate source.	
<b>scasb, ( -- )</b>	asm.sf
Compile a scasb assembler instruction.	
<b>scasd, ( -- )</b>	asm.sf
Compile a scasd assembler instruction.	
<b>scasw, ( -- )</b>	asm.sf
Compile a scasw assembler instruction.	
<b>segment ( stack-diagram -- 1st )</b>	asm.sf
When used in a stack diagram, specifies an input or output parameter with data type segment.	
<b>seta, ( mode -- )</b>	asm.sf
Compile a seta assembler instruction with mode being the destination operand.	
<b>setae, ( mode -- )</b>	asm.sf
Compile a setae assembler instruction with mode being the destination operand.	
<b>setb, ( mode -- )</b>	asm.sf
Compile a setb assembler instruction with mode being the destination operand.	
<b>setbe, ( mode -- )</b>	asm.sf
Compile a setbe assembler instruction with mode being the destination operand.	
<b>setc, ( mode -- )</b>	asm.sf
Compile a setc assembler instruction with mode being the destination operand.	
<b>sete, ( mode -- )</b>	asm.sf
Compile a sete assembler instruction with mode being the destination operand.	
<b>setg, ( mode -- )</b>	asm.sf
Compile a setg assembler instruction with mode being the destination operand.	



<b>setge, ( mode -- )</b>	asm.sf
Compile a setge assembler instruction with mode being the destination operand.	
<b>setl, ( mode -- )</b>	asm.sf
Compile a setl assembler instruction with mode being the destination operand.	
<b>setle, ( mode -- )</b>	asm.sf
Compile a setle assembler instruction with mode being the destination operand.	
<b>setna, ( mode -- )</b>	asm.sf
Compile a setna assembler instruction with mode being the destination operand.	
<b>setnae, ( mode -- )</b>	asm.sf
Compile a setnae assembler instruction with mode being the destination operand.	
<b>setnb, ( mode -- )</b>	asm.sf
Compile a setnb assembler instruction with mode being the destination operand.	
<b>setnbe, ( mode -- )</b>	asm.sf
Compile a setnbe assembler instruction with mode being the destination operand.	
<b>setnc, ( mode -- )</b>	asm.sf
Compile a setnc assembler instruction with mode being the destination operand.	
<b>setne, ( mode -- )</b>	asm.sf
Compile a setne assembler instruction with mode being the destination operand.	
<b>setng, ( mode -- )</b>	asm.sf
Compile a setng assembler instruction with mode being the destination operand.	
<b>setnge, ( mode -- )</b>	asm.sf
Compile a setnge assembler instruction with mode being the destination operand.	
<b>setnl, ( mode -- )</b>	asm.sf
Compile a setnl assembler instruction with mode being the destination operand.	
<b>setnle, ( mode -- )</b>	asm.sf

Compile a `setnle` assembler instruction with `mode` being the destination operand.

**`setno, ( mode -- )`**

asm.sf

Compile a `setno` assembler instruction with `mode` being the destination operand.

**`setnp, ( mode -- )`**

asm.sf

Compile a `setnp` assembler instruction with `mode` being the destination operand.

**`setns, ( mode -- )`**

asm.sf

Compile a `setns` assembler instruction with `mode` being the destination operand.

**`setnz, ( mode -- )`**

asm.sf

Compile a `setnz` assembler instruction with `mode` being the destination operand.

**`seto, ( mode -- )`**

asm.sf

Compile a `seto` assembler instruction with `mode` being the destination operand.

**`setp, ( mode -- )`**

asm.sf

Compile a `setp` assembler instruction with `mode` being the destination operand.

**`setpe, ( mode -- )`**

asm.sf

Compile a `setpe` assembler instruction with `mode` being the destination operand.

**`setpo, ( mode -- )`**

asm.sf

Compile a `setpo` assembler instruction with `mode` being the destination operand.

**`sets, ( mode -- )`**

asm.sf

Compile a `sets` assembler instruction with `mode` being the destination operand.

**`setz, ( mode -- )`**

asm.sf

Compile a `setz` assembler instruction with `mode` being the destination operand.

**`sgdt, ( mode -- )`**

asm.sf

Compile a `sgdt` assembler instruction with `mode` being the destination operand.

**`shl, ( mode mode -- )`**

asm.sf

Compile a `shl` assembler instruction with the first mode being the destination and the second mode being register `cl` as the count operand. An exception is thrown if the second mode is not register `cl`.

**`shl, ( mode unsigned -- )`** asm.sf

Compile a `shl` assembler instruction with mode being the destination and `unsigned` being the immediate count operand.

**`shld, ( mode mode mode -- )`** asm.sf

Compile a `shld` assembler instruction with the first mode being the destination, the second mode being the source and the third mode being register `cl` as the count operand. An exception is thrown if the second mode is not register `cl`.

**`shld, ( mode mode unsigned -- )`** asm.sf

Compile a `shld` assembler instruction with the first mode being the destination, the second mode being the source and `unsigned` being the immediate count operand.

**`shr, ( mode mode -- )`** asm.sf

Compile a `shr` assembler instruction with the first mode being the destination and the second mode being register `cl` as the count operand. An exception is thrown if the second mode is not register `cl`.

**`shr, ( mode unsigned -- )`** asm.sf

Compile a `shr` assembler instruction with mode being the destination and `unsigned` being the immediate count operand.

**`shrd, ( mode mode mode -- )`** asm.sf

Compile a `shrd` assembler instruction with the first mode being the destination, the second mode being the source and the third mode being register `cl` as the count operand. An exception is thrown if the second mode is not register `cl`.

**`shrd, ( mode mode unsigned -- )`** asm.sf

Compile a `shrd` assembler instruction with the first mode being the destination, the second mode being the source and `unsigned` being the immediate count operand.

**`si ( -- mode )`** asm.sf

mode is the register direct `si` addressing mode.

**`sidt, ( mode -- )`** asm.sf

Compile a `sidt` assembler instruction with mode being the destination operand.

**sldt, ( mode -- )** asm.sf

Compile a `sldt` assembler instruction with `mode` being the destination operand.

**smsw, ( mode -- )** asm.sf

Compile a `smsw` assembler instruction with `mode` being the destination operand.

**sp ( -- mode )** asm.sf

`mode` is the register direct `sp` addressing mode.

**sreg ( stack-diagram -- 1st )** asm.sf

When used in a stack diagram, specifies an input or output parameter with data type `sreg`.

**ss ( -- sreg )** asm.sf

`sreg` is segment register `ss`.

**ss: ( -- )** asm.sf

Compile a `ss` segment override instruction prefix.

**st ( -- mode )** asm.sf

`mode` is the floating point stack register direct `st0` addressing mode.

**st0 ( -- mode )** asm.sf

`mode` is the floating point stack register direct `st0` addressing mode.

**st1 ( -- mode )** asm.sf

`mode` is the floating point stack register direct `st1` addressing mode.

**st2 ( -- mode )** asm.sf

`mode` is the floating point stack register direct `st2` addressing mode.

**st3 ( -- mode )** asm.sf

`mode` is the floating point stack register direct `st3` addressing mode.

**st4 ( -- mode )** asm.sf

`mode` is the floating point stack register direct `st4` addressing mode.

<b>st5 ( -- mode )</b>	asm.sf
mode is the floating point stack register direct st5 addressing mode.	
<b>st6 ( -- mode )</b>	asm.sf
mode is the floating point stack register direct st6 addressing mode.	
<b>st7 ( -- mode )</b>	asm.sf
mode is the floating point stack register direct st7 addressing mode.	
<b>stc, ( -- )</b>	asm.sf
Compile a stc assembler instruction.	
<b>std, ( -- )</b>	asm.sf
Compile a std assembler instruction.	
<b>sti, ( -- )</b>	asm.sf
Compile a sti assembler instruction.	
<b>stosb, ( -- )</b>	asm.sf
Compile a stosb assembler instruction.	
<b>stosd, ( -- )</b>	asm.sf
Compile a stosd assembler instruction.	
<b>stosw, ( -- )</b>	asm.sf
Compile a stosw assembler instruction.	
<b>str, ( mode -- )</b>	asm.sf
Compile a str assembler instruction with mode being the destination operand.	
<b>sub, ( mode mode -- )</b>	asm.sf
Compile a sub assembler instruction with the first mode being the destination and the second mode being the source.	
<b>sub, ( mode single -- )</b>	asm.sf
Compile a sub assembler instruction with mode being the destination and single being the immediate source.	

<b>syscall, ( -- )</b>	asm.sf
Compile a <code>syscall</code> assembler instruction.	
<b>sysenter, ( -- )</b>	asm.sf
Compile a <code>sysenter</code> assembler instruction.	
<b>sysexit, ( -- )</b>	asm.sf
Compile a <code>sysexit</code> assembler instruction.	
<b>sysret, ( -- )</b>	asm.sf
Compile a <code>sysret</code> assembler instruction.	
<b>test, ( mode mode -- )</b>	asm.sf
Compile a <code>test</code> assembler instruction with the first <code>mode</code> being the first source operand and the second <code>mode</code> being the second source operand.	
<b>test, ( mode single -- )</b>	asm.sf
Compile a <code>test</code> assembler instruction with <code>mode</code> being the first source operand and <code>single</code> being the second (immediate) source operand.	
<b>then, ( origin-address -- )</b>	asm.sf
Resolve a forward jump by compiling the jump offset from <code>origin-address</code> to the current position of the code space.	
<b>tzcnt, ( mode mode -- )</b>	asm.sf
Compile a <code>tzcnt</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being the source.	
<b>ud2, ( -- )</b>	asm.sf
Compile an <code>ud2</code> assembler instruction.	
<b>unchanged ( stack-diagram -- 1st )</b>	
During specification of the stack diagram <code>stack-diagram</code> of a definition, mark a list of registers as being preserved by the definition. The preservation is guaranteed by compiling appropriate <code>push</code> , an <code>pop</code> , assembler instructions at the beginning and at the end of the definition, respectively. <code>1st</code> is equal to <code>stack-diagram</code> . Registers have to be considered only when programming in assembler.	

<b>untila, ( destination-address -- )</b>	asm.sf
Compile a jna assembler instruction with destination-address being the jump destination.	
<b>untilae, ( destination-address -- )</b>	asm.sf
Compile a jnae assembler instruction with destination-address being the jump destination.	
<b>untilb, ( destination-address -- )</b>	asm.sf
Compile a jnb assembler instruction with destination-address being the jump destination.	
<b>untilbe, ( destination-address -- )</b>	asm.sf
Compile a jnbe assembler instruction with destination-address being the jump destination.	
<b>untilc, ( destination-address -- )</b>	asm.sf
Compile a jnc assembler instruction with destination-address being the jump destination.	
<b>untile, ( destination-address -- )</b>	asm.sf
Compile a jne assembler instruction with destination-address being the jump destination.	
<b>untilg, ( destination-address -- )</b>	asm.sf
Compile a jng assembler instruction with destination-address being the jump destination.	
<b>untilge, ( destination-address -- )</b>	asm.sf
Compile a jnge assembler instruction with destination-address being the jump destination.	
<b>untill, ( destination-address -- )</b>	asm.sf
Compile a jnl assembler instruction with destination-address being the jump destination.	
<b>untille, ( destination-address -- )</b>	asm.sf
Compile a jnle assembler instruction with destination-address being the jump destination.	
<b>untilna, ( destination-address -- )</b>	asm.sf
Compile a ja assembler instruction with destination-address being the jump destination.	
<b>untilnae, ( destination-address -- )</b>	asm.sf

Compile a `jae` assembler instruction with `destination-address` being the jump destination.

**untilnb, ( destination-address -- )** asm.sf

Compile a `jnb` assembler instruction with `destination-address` being the jump destination.

**untilnbe, ( destination-address -- )** asm.sf

Compile a `jbe` assembler instruction with `destination-address` being the jump destination.

**untilnc, ( destination-address -- )** asm.sf

Compile a `jc` assembler instruction with `destination-address` being the jump destination.

**untilncxz, ( destination-address -- )** asm.sf

Compile a `jcxz` assembler instruction with `destination-address` being the jump destination.

**untilne, ( destination-address -- )** asm.sf

Compile a `je` assembler instruction with `destination-address` being the jump destination.

**untilnecxz, ( destination-address -- )** asm.sf

Compile a `jecxz` assembler instruction with `destination-address` being the jump destination.

**untilng, ( destination-address -- )** asm.sf

Compile a `jg` assembler instruction with `destination-address` being the jump destination.

**untilnge, ( destination-address -- )** asm.sf

Compile a `jge` assembler instruction with `destination-address` being the jump destination.

**untilnl, ( destination-address -- )** asm.sf

Compile a `jnl` assembler instruction with `destination-address` being the jump destination.

**untilnle, ( destination-address -- )** asm.sf

Compile a `jle` assembler instruction with `destination-address` being the jump destination.

**untilno, ( destination-address -- )** asm.sf

Compile a `jno` assembler instruction with `destination-address` being the jump destination.



<b>untilnp, ( destination-address -- )</b>	asm.sf
Compile a <code>jmp</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilns, ( destination-address -- )</b>	asm.sf
Compile a <code>jns</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilnz, ( destination-address -- )</b>	asm.sf
Compile a <code>jnz</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilno, ( destination-address -- )</b>	asm.sf
Compile a <code>jno</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilp, ( destination-address -- )</b>	asm.sf
Compile a <code>jnp</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilpe, ( destination-address -- )</b>	asm.sf
Compile a <code>jpo</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilpo, ( destination-address -- )</b>	asm.sf
Compile a <code>jpe</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untils, ( destination-address -- )</b>	asm.sf
Compile a <code>jns</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>untilz, ( destination-address -- )</b>	asm.sf
Compile a <code>jnz</code> assembler instruction with <code>destination-address</code> being the jump destination.	
<b>verr, ( mode -- )</b>	asm.sf
Compile a <code>verr</code> assembler instruction with <code>mode</code> being the source operand.	
<b>verw, ( mode -- )</b>	asm.sf
Compile a <code>verw</code> assembler instruction with <code>mode</code> being the source operand.	
<b>wait, ( -- )</b>	asm.sf
Compile a <code>wait</code> assembler instruction.	
<b>wbinvd, ( -- )</b>	asm.sf

Compile a `wbinvd` assembler instruction.

**whilea, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jna` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whileae, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnae` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whileb, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnb` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilebe, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnbe` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilec, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnc` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilee, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jne` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whileg, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jng` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilege, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnge` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilel, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnl` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilele, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnl` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilena, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `ja` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilenae, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jae` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilenb, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jb` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilenbe, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jbe` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilenc, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jc` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilencxz, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jcxz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilene, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `je` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whilenecxz, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jecxz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**`whileng, ( destination-address -- origin-address 1st )`** asm.sf

Compile a `jg` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilenge, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jge` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilenl, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnl` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilenle, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jle` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whileno, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jno` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilenp, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnp` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilens, ( destination-address -- origin-address 1st )** asm.sf

Compile a `js` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilenz, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jz` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whileo, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jno` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilep, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jnp` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

**whilepe, ( destination-address -- origin-address 1st )** asm.sf

Compile a `jpo` assembler instruction with a dummy jump destination. `origin-address` is a pointer to the code space location after this instruction. `1st` is `destination-address`.

<b>whilepo, ( destination-address -- origin-address 1st )</b>	asm.sf
Compile a <code>jpe</code> assembler instruction with a dummy jump destination. <code>origin-address</code> is a pointer to the code space location after this instruction. <code>1st</code> is <code>destination-address</code> .	
<b>whiles, ( destination-address -- origin-address 1st )</b>	asm.sf
Compile a <code>jns</code> assembler instruction with a dummy jump destination. <code>origin-address</code> is a pointer to the code space location after this instruction. <code>1st</code> is <code>destination-address</code> .	
<b>whilez, ( destination-address -- origin-address 1st )</b>	asm.sf
Compile a <code>jnz</code> assembler instruction with a dummy jump destination. <code>origin-address</code> is a pointer to the code space location after this instruction. <code>1st</code> is <code>destination-address</code> .	
<b>wrmsr, ( -- )</b>	asm.sf
Compile a <code>wrmsr</code> assembler instruction.	
<b>xadd, ( mode mode -- )</b>	asm.sf
Compile a <code>xadd</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being the source.	
<b>xchg, ( mode mode -- )</b>	asm.sf
Compile a <code>xchg</code> assembler instruction with the first <code>mode</code> being the first operand and the second <code>mode</code> being the second operand.	
<b>xlat, ( -- )</b>	asm.sf
Compile a <code>xlat</code> assembler instruction.	
<b>xlatb, ( -- )</b>	asm.sf
Compile a <code>xlatb</code> assembler instruction.	
<b>xor, ( mode mode -- )</b>	asm.sf
Compile a <code>xor</code> assembler instruction with the first <code>mode</code> being the destination and the second <code>mode</code> being the source.	
<b>xor, ( mode single -- )</b>	asm.sf
Compile a <code>xor</code> assembler instruction with <code>mode</code> being the destination and <code>single</code> being the immediate source.	
<b>[eax] ( -- mode )</b>	asm.sf
<code>mode</code> is the register indirect <code>[eax]</code> addressing mode.	

**[eax]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [eax]+disp addressing mode. single is the displacement.

**[ebp] ( -- mode )** asm.sf

mode is the register indirect [ebp] addressing mode.

**[ebp]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [ebp]+disp addressing mode for. single is the displacement.

**[ebx] ( -- mode )** asm.sf

mode is the register indirect [ebx] addressing mode.

**[ebx]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [ebx]+disp addressing mode. single is the displacement.

**[ecx] ( -- mode )** asm.sf

mode is the register indirect [ecx] addressing mode.

**[ecx]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [ecx]+disp addressing mode. single is the displacement.

**[edi] ( -- mode )** asm.sf

mode is the register indirect [edi] addressing mode.

**[edi]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [edi]+disp addressing mode. single is the displacement.

**[edx] ( -- mode )** asm.sf

mode is the register indirect [edx] addressing mode.

**[edx]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [edx]+disp addressing mode. single is the displacement.

**[esi] ( -- mode )** asm.sf

mode is the register indirect [esi] addressing mode.

**[esi]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [esi]+disp addressing mode. single is the displacement.

**[esp] ( -- mode )** asm.sf

mode is the register indirect [esp] addressing mode.

**[esp]+ ( single -- mode )** asm.sf

mode is the register indirect with displacement [esp]+disp addressing mode. single is the displacement.

**[] ( single -- mode )** asm.sf

mode is the indirect addressing mode. single is the memory address.