The amsmath package

Frank Mittelbach Rainer Schöpf Michael Downes
David M. Jones

Version 2.14, 2013/01/14

1 Introduction

A LaTeX package named amstex was created in 1988–1989 by adapting amstex.tex for use within LaTeX. The amsmath package is the successor of the amstex package. It was substantially overhauled to integrate it with LaTeX2e, which arrived on the scene in 1994. It provides more or less the same features, but there are quite a few organizational differences as well as some new features and bug fixes. For example, the amstex package automatically loaded the amsfonts package, but the amsmath package does not. At the present time (November 1999) user-level documentation of the commands provided here is found in the AMSmath Users' Guide, amsldoc.tex.

Standard file identification.

```
\NeedsTeXFormat{LaTeX2e}% LaTeX 2.09 can't be used (nor non-LaTeX)
[1994/12/01]% LaTeX date must be December 1994 or later
\ProvidesPackage{amsmath}[2013/01/14 v2.14 AMS math features]
```

2 Catcode defenses

Some packages change the catcode of characters that are essential in low-level TeX syntax. Any package that does so does not qualify as a PWWO package ("Plays Well With Others") because it can cause other packages to fail if they are loaded later. LaTeX is partly to blame for this because it fails to provide adequate built-in safeguards in the package loading mechanisms. In the absence of such safeguards, we will provide them here.

```
\edef\@temp{\catcode 96=\number\catcode 96 }
\catcode\string `\`=12
\def\do#1{\catcode\number`#1=\number\catcode`#1}
\edef\@temp{%
  \noexpand\AtEndOfPackage{%
    \@temp
    \do\"\do\'\do\(\do\)\do\*\do\+\do\,\do\-\do\.%
    \do\/\do\<\do\=\do\>\do\[\do\]\do\^\do\_\relax
  }%
}
\@temp
\def\do#1{\catcode\number`#1=12 }
\do\"\do\'\do\(\do\)\do\*\do\+\do\,\do\-\do\.
\do\/\do\<\do\=\do\>\do\[\do\]
\catcode`\^=7 \catcode`\_=8
```

3 Declare some options

Handling of limits on integrals, sums, operatornames.

```
\DeclareOption{intlimits}{\let\ilimits@\displaylimits}
\DeclareOption{nointlimits}{\let\ilimits@\nolimits}
```

```
\DeclareOption{sumlimits}{\let\slimits@\displaylimits}
\DeclareOption{nosumlimits}{\let\slimits@\nolimits}
\DeclareOption{namelimits}{\PassOptionsToPackage{namelimits}{amsopn}}
\DeclareOption{nonamelimits}{%
  \PassOptionsToPackage{nonamelimits}{amsopn}}
```

The following two switches might have been defined already by the documentclass, but it doesn't hurt to re-execute the `\newif`'s.

```
\newif\ifctagsplit@
\newif\iftagsleft@
```

Right or left placement of equation numbers.

```
\DeclareOption{leqno}{\tagsleft@true}
\DeclareOption{reqno}{\tagsleft@false}
\DeclareOption{centertags}{\ctagsplit@true}
\DeclareOption{tbtags}{\ctagsplit@false}
```

The `cmex10` option is an escape hatch for people who don't happen to have sizes 7–9 of the `cmex` fonts available to them yet. (Strictly speaking they are considered part of a minimum LaTeX distribution now, i.e., all LaTeX 2_ε users should have them, without needing to get the AMSFonts distrib.)

```
\DeclareOption{cmex10}{%
  \ifnum\cmex@opt=\@ne \def\cmex@opt{0}%
  \else \def\cmex@opt{10}\fi
}
```

To help things work out better with various package loading orders of `amsmath` and `amsfonts`, we establish a variable to communicate the status of the cmex font definition. If the `amsfonts` package was loaded first this variable might be already defined, in which case we want to preserve its value.

```
\@ifundefined{cmex@opt}{\def\cmex@opt{7}}{}
```

4 Flush-left equations [DMJ]

The left margin of math enviroments is controlled by `\@mathmargin`. This can be set to `\@centering` to implement the default behaviour, i.e., centered equations, and to something else to implement the flushleft style.

In theory, all that's needed to activate the flushleft mode in the AMS document classes is something like this:

```
\DeclareOption{fleqn}{%
  \AtBeginDocument{\@mathmargin30pt\relax}%
}
```

(In fact, unless the document class wants to specify the `\@mathmargin`, it doesn't need to do anything with the `fleqn` option.)

```
\newif\if@fleqn
%
\newskip\@mathmargin
\@mathmargin\@centering
%
\DeclareOption{fleqn}{%
    \@fleqntrue
    \@mathmargin = -1sp
    \let\mathindent=\@mathmargin
    \AtBeginDocument{%
        \ifdim\@mathmargin= -1sp
            \@mathmargin\leftmargini minus\leftmargini
        \fi
    }%
}
```

DMJ: This ensures that `\@mathmargin` is given some sort of sensible default if the class doesn't specify one, while still allowing a user to override the default value in their document preamble. (Incidentally, I'm initializing `\@mathmargin` to `\leftmargini` for compatibility with `fleqn.clo`, but I'm not at all convinced that's the right thing to do.)

The next question is what happens when amsmath is used with one of the standard classes. Unfortunately, LaTeX implements `fleqn` somewhat clumsily; instead of paramaterizing the definitions of the math structures (as I've attempted to do here), `fleqn.clo` declares a dimen `\mathindent` that is much like my `\@mathmargin` and then redefines `\[`, `\]`, `\equation`, and `\eqnarray`. This means that things could get rather messy in 2.09 compatibility mode, since `fleqn.clo` might be loaded after `amsmath.sty`, which could cause a real mess.

[mjd,1999/07/07]: Let `\mathindent` = `\@mathmargin` as envisioned by DMJ. Compatibility-mode documents will all use the `amstex` package, not `amsmath`. There is a remote chance of a problem if someone makes an assignment to `\mathindent` in a way that implicitly assumes it is a dimen register (inasmuch as it has now become a skip register), and the string "plus" follows in the input stream, but if someone's document croaks in that way, I think they will just have to bite the bullet and fix it. The alternative is to penalize a lot of other users with a known handicap.

```
\DeclareOption{?}{}
\ExecuteOptions{nointlimits,sumlimits,namelimits,centertags}
```

The `\par` after `\ProcessOptions` is to ensure the correct line number on screen if an error occurs during option processing; otherwise the lookahead for a `*` option would result in TeX showing the following line instead.

```
\ProcessOptions\par
\@ifpackagewith{amsmath}{?}{%
  \typeout{^^J%
Documentation for the amsmath package is found in amsldoc.dvi^^J%
(or .pdf or .tex).^^J%
^^J%
See also http://www.ams.org/tex/amslatex.html.^^J%
^^J%
Note: Using the first edition of The LaTeX Companion (1994) without^^J%
errata as a guide for amsmath use is not recommended.^^J%
  }%
}{%
  \typeout{%
For additional information on amsmath, use the \lq ?\rq\space option.%
  }%
}
```

Processing to handle the `cmex10` option is a little tricky because of different possible loading orders for `amsmath` and `amsfonts`. The package `amsmath` sets the `\cmex@opt` flag to 7 or 10, and the package `amsfonts` sets the flag to 1 or 0.

```
\ifnum\cmex@opt=7 \relax
  \DeclareFontShape{OMX}{cmex}{m}{n}{%
    <-8>cmex7<8>cmex8<9>cmex9%
    <10><10.95><12><14.4><17.28><20.74><24.88>cmex10%
  }{}%
  \expandafter\let\csname OMX/cmex/m/n/10\endcsname\relax
\else
  \ifnum\cmex@opt=\z@ % need to override cmex7 fontdef from amsfonts
```

Force reloading of the OMX/cmex font definition file.

```
    \begingroup
```

```
    \fontencoding{OMX}\fontfamily{cmex}%
    \expandafter\let\csname OMX+cmex\endcsname\relax
    \try@load@fontshape
    \endgroup
```

The `cmex10` font gets special preload handling in the building of the LaTeX format file, need an extra bit here to work around that.

```
    \expandafter\let\csname OMX/cmex/m/n/10\endcsname\relax
    \def\cmex@opt{10}%
  \fi
\fi
```

5 Call some other packages

The `amstext` package provides the `\text` command. The `amsbsy` package provides `\boldsymbol` and `\pmb`. (Since 1997 it is usually better to use the `bm` package instead; but I think we have to keep `amsbsy` here for backward compatibility [mjd,1999/11/19].) The `amsopn` package provides `\DeclareMathOperator`.

```
\RequirePackage{amstext}[1995/01/25]
\RequirePackage{amsbsy}[1995/01/20]
\RequirePackage{amsopn}[1995/01/20]
```

6 Miscellaneous

`\ams@newcommand` Where `stix` and `amsmath` define the same control sequences, we want to avoid inadvertently overriding `stix`'s definitions. If `stix` is loaded before `amsmath`, the following conditional takes care of the problem. There is similar code in the `stix` package in case `amsmath` is loaded first.

```
\@ifpackageloaded{stix}{%
    \let\ams@newcommand\providecommand
    \let\ams@renewcommand\providecommand
    \let\ams@def\providecommand
    \let\ams@DeclareRobustCommand\providecommand
}{%
    \let\ams@newcommand\newcommand
    \let\ams@renewcommand\renewcommand
    \let\ams@def\def
    \let\ams@DeclareRobustCommand\DeclareRobustCommand
}
```

`\@amsmath@err` Defining this error function saves main mem.

```
\def\@amsmath@err{\PackageError{amsmath}}
```

`\AmS` The `\AmS` prefix can be used to construct the combination `\AmS-\LaTeX`.

```
\providecommand{\AmS}{{\protect\AmSfont
  A\kern-.1667em\lower.5ex\hbox{M}\kern-.125emS}}
```

In `\AmSfont` we call cmsy directly in lieu of trying to access it through the math fonts setup (e.g. `\the\textfont2`) because math fonts can't be relied on to be properly set up if we are not inside a math formula. This means that if this command is used in a document where CM fonts are not wanted, then a font substitution will need to be declared, e.g.:

```
\DeclareFontShape{OMS}{cmsy}{m}{n}{ <-> sub * xxx/m/n }{}
```

where `xxx` is some alternate font family. Taking the first letter of `\f@series` will produce `b` or `m` for the most common values (`b,bx,m`). It may produce nonsense for more unusual values of `\f@series`, so for safety's sake we have an additional

\if test. We want to avoid setting the series to bx because in a standard LaTeX installation the combination cmsy/bx/n does not have a font definition, and the user would get a font substitution warning on screen.

```
\newcommand{\AmSfont}{%
  \usefont{OMS}{cmsy}{\if\@xp\@car\f@series\@nil bb\else m\fi}{n}}
```

\@mathmeasure The function \@mathmeasure takes three arguments; the third arg is typeset as a math formula in an hbox, using arg #2 as the mathstyle, and the result is left in the box named by the first arg. It is assumed that we are already in math mode, so we can turn off \everymath (in particular, \check@mathfonts).

```
\def\@mathmeasure#1#2#3{\setbox#1\hbox{\frozen@everymath\@emptytoks
  \m@th$#2#3$}}
```

The \inf@bad constant is for testing overfull boxes.

```
\@ifundefined{inf@bad}{%
  \newcount\inf@bad \inf@bad=1000000 \relax
}{}
```

6.1 Math spacing commands

Here we fill in some gaps in the set of spacing commands, and make them all work equally well in or out of math. We want all these commands to be robust but declaring them all with \DeclareRobustCommand uses up an control sequence name per command; to avoid this, we define a common command \tmspace (text-or-math space) which carries the robustness burden for all of them. The standard \relax before the \ifmmode is not necessary because of the \protect added by \DeclareRobustCommand.

```
\DeclareRobustCommand{\tmspace}[3]{%
  \ifmmode\mskip#1#2\else\kern#1#3\fi\relax}
\renewcommand{\,}{\tmspace+\thinmuskip{.1667em}}
\let\thinspace\,
\renewcommand{\!}{\tmspace-\thinmuskip{.1667em}}
\let\negthinspace\!
\renewcommand{\:}{\tmspace+\medmuskip{.2222em}}
\let\medspace\:
\newcommand{\negmedspace}{\tmspace-\medmuskip{.2222em}}
\renewcommand{\;}{\tmspace+\thickmuskip{.2777em}}
\let\thickspace\;
\newcommand{\negthickspace}{\tmspace-\thickmuskip{.2777em}}
```

\mspace And while we're at it, why don't we provide an equivalent of \hspace for math mode use. This allows use of mu units in (for example) constructing compound math symbols.

```
\newcommand{\mspace}[1]{\mskip#1\relax}
```

6.2 Vertical bar symbols

Add left/right specific versions of \vert, \Vert. Don't assume the delimiter codes are the CM defaults.

```
\def\@tempa#1#2\@nil{%
  \ifx\delimiter#1\@tempcnta#2\relax\else\@tempcnta\z@\fi
}
\@xp\@tempa\vert\@empty\@nil
\ifnum\@tempcnta>\z@
  \advance\@tempcnta "4000000
  \xdef\lvert{\delimiter\number\@tempcnta\space }
  \advance\@tempcnta "1000000
  \xdef\rvert{\delimiter\number\@tempcnta\space }
\else
```

```
  \ifx\@@undefined\lvert
    % Fall back to cmex encoding since we don't know what else to do.
    \DeclareMathDelimiter{\lvert}
      {\mathopen}{symbols}{"6A}{largesymbols}{"0C}
    \DeclareMathDelimiter{\rvert}
      {\mathclose}{symbols}{"6A}{largesymbols}{"0C}
  \fi
\fi
\@xp\@tempa\Vert\@empty\@nil
\ifnum\@tempcnta>\z@
  \advance\@tempcnta "4000000
  \xdef\lVert{\delimiter\number\@tempcnta\space }
  \advance\@tempcnta "1000000
  \xdef\rVert{\delimiter\number\@tempcnta\space }
\else
  \ifx\@@undefined\lVert
    \DeclareMathDelimiter{\lVert}
      {\mathopen}{symbols}{"6B}{largesymbols}{"0D}
    \DeclareMathDelimiter{\rVert}
      {\mathclose}{symbols}{"6B}{largesymbols}{"0D}
  \fi
\fi
```

6.3 Fractions

Bury the generalized fraction primitives `\over`, `\atop`, etc., because of their bizarre syntax, which is decidedly out of place in a LaTeX document.

```
\@saveprimitive\over\@@over
\@saveprimitive\atop\@@atop
\@saveprimitive\above\@@above
\@saveprimitive\overwithdelims\@@overwithdelims
\@saveprimitive\atopwithdelims\@@atopwithdelims
\@saveprimitive\abovewithdelims\@@abovewithdelims
```

If someone insists on using `\over`, give a warning the first time and then resurrect the old definition. Laissez-faire policy.

```
\DeclareRobustCommand{\primfrac}[1]{%
  \PackageWarning{amsmath}{%
Foreign command \@backslashchar#1;\MessageBreak
\protect\frac\space or \protect\genfrac\space should be used instead%
\MessageBreak
  }
  \global\@xp\let\csname#1\@xp\endcsname\csname @@#1\endcsname
  \csname#1\endcsname
}
\renewcommand{\over}{\primfrac{over}}
\renewcommand{\atop}{\primfrac{atop}}
\renewcommand{\above}{\primfrac{above}}
\renewcommand{\overwithdelims}{\primfrac{overwithdelims}}
\renewcommand{\atopwithdelims}{\primfrac{atopwithdelims}}
\renewcommand{\abovewithdelims}{\primfrac{abovewithdelims}}
```

`\frac` calls `\@@over` directly instead of via `\genfrac`, for better speed because it is so common. `\tfrac` and `\dfrac` are abbreviations for some commonly needed mathstyle overrides. To conserve csnames we avoid making `\dfrac` and `\tfrac` robust (`\genfrac` is itself robust).

```
\DeclareRobustCommand{\frac}[2]{{\begingroup#1\endgroup\@@over#2}}
\newcommand{\dfrac}{\genfrac{}{}{}0}
\newcommand{\tfrac}{\genfrac{}{}{}1}
```

The \binom command for binomial notation works like \frac and has similar variants. Note that we do not use \z@ in \dbinom and \tbinom because they are not top-level robust like \binom, and so the \z@ with the potentially problematic @ character would become visible when writing one of those commands to a .toc file.

```
\DeclareRobustCommand{\binom}{\genfrac()\z@{}}
\newcommand{\dbinom}{\genfrac(){0pt}0}
\newcommand{\tbinom}{\genfrac(){0pt}1}
```

\genfrac This command provides access to TEX's generalized fraction primitives. Args: #1 left delim, #2 right delim, #3 line thickness, #4 mathstyle override, #5 numerator, #6 denominator. But we only read the first four args at first, in order to give us a moment to select the proper generalized fraction primitive. Any of those four args could be empty, and when empty the obvious defaults are selected (no delimiters, default line thickness (normally .4pt), and no mathstyle override).

```
\DeclareRobustCommand{\genfrac}[4]{%
  \def\@tempa{#1#2}%
  \edef\@tempb{\@nx\@genfrac\@mathstyle{#4}%
    \csname @@\ifx @#3@over\else above\fi
    \ifx\@tempa\@empty \else withdelims\fi\endcsname}
  \@tempb{#1#2#3}}
```

\@genfrac takes the preceding arguments and reads the numerator and denominator. Note that there's no convenient way to make the numerator and denominator *contents* displaystyle, through this interface.

Args: #1 mathstyle, #2 fraction primitive, #3 delimiters and rule thickness, #4 numerator, #5 denominator.

```
\def\@genfrac#1#2#3#4#5{{#1{\begingroup#4\endgroup#2#3\relax#5}}}
```

Empty mathstyle arg: no change; 0 = displaystyle, 1 = textstyle, 2 = scriptstyle, 3 = scriptscriptstyle.

```
\def\@mathstyle#1{%
  \ifx\@empty#1\@empty\relax
  \else\ifcase#1\displaystyle % case 0
    \or\textstyle\or\scriptstyle\else\scriptscriptstyle\fi\fi}
```

6.4 Sums and Integrals

Default value for sum limits is \displaylimits, see option 'nosumlimits'.

We redefine all the cumulative operator symbols to use \slimits@ so that switching between \displaylimits and \nolimits can be controlled by package options. Also add \DOTSB for the benefit of the dots lookahead. But we'd better make sure \coprod and the others are simple mathchars; if not, the attempted changes will probably fail miserably.

```
\begingroup
\edef\@tempa{\string\mathchar"}
\def\@tempb#1"#2\@nil{#1"}
\edef\@tempc{\expandafter\@tempb\meaning\coprod "\@nil}
\ifx\@tempa\@tempc
  \global\let\coprod@\coprod
  \gdef\coprod{\DOTSB\coprod@\slimits@}
  \global\let\bigvee@\bigvee
  \gdef\bigvee{\DOTSB\bigvee@\slimits@}
  \global\let\bigwedge@\bigwedge
  \gdef\bigwedge{\DOTSB\bigwedge@\slimits@}
  \global\let\biguplus@\biguplus
  \gdef\biguplus{\DOTSB\biguplus@\slimits@}
```

```
  \global\let\bigcap@\bigcap
  \gdef\bigcap{\DOTSB\bigcap@\slimits@}
  \global\let\bigcup@\bigcup
  \gdef\bigcup{\DOTSB\bigcup@\slimits@}
  \global\let\prod@\prod
  \gdef\prod{\DOTSB\prod@\slimits@}
  \global\let\sum@\sum
  \gdef\sum{\DOTSB\sum@\slimits@}
  \global\let\bigotimes@\bigotimes
  \gdef\bigotimes{\DOTSB\bigotimes@\slimits@}
  \global\let\bigoplus@\bigoplus
  \gdef\bigoplus{\DOTSB\bigoplus@\slimits@}
  \global\let\bigodot@\bigodot
  \gdef\bigodot{\DOTSB\bigodot@\slimits@}
  \global\let\bigsqcup@\bigsqcup
  \gdef\bigsqcup{\DOTSB\bigsqcup@\slimits@}
\fi
\endgroup
```

6.5 Roots and radicals

This root stuff needs syntax work and implementation work. Surely something more compact can be done?? [mjd, 1994/09/05]

```
\newcommand{\leftroot}{\@amsmath@err{\Invalid@@\leftroot}\@eha}
\newcommand{\uproot}{\@amsmath@err{\Invalid@@\uproot}\@eha}
\newcount\uproot@
\newcount\leftroot@
\renewcommand{\root}{\relaxnext@
  \DN@{\ifx\@let@token\uproot\let\next@\nextii@\else
   \ifx\@let@token\leftroot\let\next@\nextiii@\else
   \let\next@\plainroot@\fi\fi\next@}%
  \def\nextii@\uproot##1{\uproot@##1\relax\FN@\nextiv@}%
  \def\nextiv@{\ifx\@let@token\@sptoken\DN@. {\FN@\nextv@}\else
   \DN@.{\FN@\nextv@}\fi\next@.}%
  \def\nextv@{\ifx\@let@token\leftroot\let\next@\nextvi@\else
   \let\next@\plainroot@\fi\next@}%
  \def\nextvi@\leftroot##1{\leftroot@##1\relax\plainroot@}%
   \def\nextiii@\leftroot##1{\leftroot@##1\relax\FN@\nextvii@}%
  \def\nextvii@{\ifx\@let@token\@sptoken
   \DN@. {\FN@\nextviii@}\else
   \DN@.{\FN@\nextviii@}\fi\next@.}%
  \def\nextviii@{\ifx\@let@token\uproot\let\next@\nextix@\else
   \let\next@\plainroot@\fi\next@}%
  \def\nextix@\uproot##1{\uproot@##1\relax\plainroot@}%
  \bgroup\uproot@\z@\leftroot@\z@\FN@\next@}
\def\plainroot@#1\of#2{\setbox\rootbox\hbox{%
 $\m@th\scriptscriptstyle{#1}$}%
 \mathchoice{\r@@t\displaystyle{#2}}{\r@@t\textstyle{#2}}
 {\r@@t\scriptstyle{#2}}{\r@@t\scriptscriptstyle{#2}}\egroup}
```

Name change from `\@@sqrt` to `\sqrtsign` happened in the 1995/12/01 release of LaTeX. If we were to assume that `\sqrtsign` is defined then someone with the 1995/06/01 release of LaTeX would have trouble using this package.

```
\@ifundefined{sqrtsign}{\let\sqrtsign\@@sqrt}{}
\def\r@@t#1#2{\setboxz@h{$\m@th#1\sqrtsign{#2}$}%
 \dimen@\ht\z@\advance\dimen@-\dp\z@
 \setbox\@ne\hbox{$\m@th#1\mskip\uproot@ mu$}%
 \advance\dimen@ by1.667\wd\@ne
 \mkern-\leftroot@ mu\mkern5mu\raise.6\dimen@\copy\rootbox
 \mkern-10mu\mkern\leftroot@ mu\boxz@}
```

6.6 Et cetera

Specific names for the variant italic cap Greek letters are not defined by LaTeX. If no preceding package defined these, we will define them now.

```
\@ifundefined{varGamma}{%
  \DeclareMathSymbol{\varGamma}{\mathord}{letters}{"00}
  \DeclareMathSymbol{\varDelta}{\mathord}{letters}{"01}
  \DeclareMathSymbol{\varTheta}{\mathord}{letters}{"02}
  \DeclareMathSymbol{\varLambda}{\mathord}{letters}{"03}
  \DeclareMathSymbol{\varXi}{\mathord}{letters}{"04}
  \DeclareMathSymbol{\varPi}{\mathord}{letters}{"05}
  \DeclareMathSymbol{\varSigma}{\mathord}{letters}{"06}
  \DeclareMathSymbol{\varUpsilon}{\mathord}{letters}{"07}
  \DeclareMathSymbol{\varPhi}{\mathord}{letters}{"08}
  \DeclareMathSymbol{\varPsi}{\mathord}{letters}{"09}
  \DeclareMathSymbol{\varOmega}{\mathord}{letters}{"0A}
}{}
```

$\mathcal{AMS}$-TeX redefines `\overline` as shown here, for reasons that are probably less important in LaTeX: Make it read its argument as a macro argument rather than a "math field" (*The TeXbook*, Chapter 26), to avoid problems when something that is apparently a single symbol is actually a non-simple macro (e.g., `\dag`) *and* is given as a single-token argument without enclosing braces.

```
\@saveprimitive\overline\@@overline
\DeclareRobustCommand{\overline}[1]{\@@overline{#1}}
```

The `\boxed` command is specifically intended to put a box around an equation or piece of an equation. (Not including the equation number.) This isn't trivial for end-users to do it properly with `\fbox` so we provide a command for them.

```
\newcommand{\boxed}[1]{\fbox{\m@th$\displaystyle#1$}}
```

```
\newcommand{\implies}{\DOTSB\;\Longrightarrow\;}
\newcommand{\impliedby}{\DOTSB\;\Longleftarrow\;}
```

```
\def\And{\DOTSB\;\mathchar"3026 \;}
```

The command `\nobreakdash` is designed only for use before a hyphen or dash (`-`, `--`, or `---`). Setting the hyphen in a box and then unboxing it means that the normal penalty will not be added after it—and if the penalty is not there a break will not be taken (unless an explicit penalty or glue follows, thus the final `\nobreak`).

```
\newcommand{\nobreakdash}{\leavevmode
  \toks@\@emptytoks \def\@tempa##1{\toks@\@xp{\the\toks@-}\FN@\next@}%
  \DN@{\ifx\@let@token-\@xp\@tempa
       \else\setboxz@h{\the\toks@\nobreak}\unhbox\z@\fi}%
  \FN@\next@
}
```

`\colon` is for a colon in math that resembles a text colon: small space on the left, larger space on the right. The `:` character by itself is treated as a `\mathrel` i.e. large, equal spacing on both sides.

```
\renewcommand{\colon}{\nobreak\mskip2mu\mathpunct{}\nonscript
  \mkern-\thinmuskip{:}\mskip6muplus1mu\relax}
```

7 Ellipsis dots

We can't use `\newif` for `\ifgtest@` because we want to include `\global` in the definitions of `\gtest@true` and `\gtest@false`.

```
\let\ifgtest@\iffalse                              % initial value
\def\gtest@true{\global\let\ifgtest@\iftrue}
```

```
\def\gtest@false{\global\let\ifgtest@\iffalse}
\let\DOTSI\relax
\let\DOTSB\relax
\let\DOTSX\relax
{\uccode`7=`\\ \uccode`8=`m \uccode`9=`a \uccode`0=`t \uccode`!=`h
 \uppercase{%
  \gdef\math@#1#2#3#4#5#6\math@{\gtest@false\ifx 7#1\ifx 8#2%
  \ifx 9#3\ifx 0#4\ifx !#5\xdef\meaning@{#6}\gtest@true
  \fi\fi\fi\fi\fi}}}
{\uccode`7=`c \uccode`8=`h \uccode`9=`\"
 \uppercase{\gdef\mathch@#1#2#3#4#5#6\mathch@{\gtest@false
  \ifx 7#1\ifx 8#2\ifx 9#5\gtest@true\xdef\meaning@{9#6}\fi\fi\fi}}}
\newcount\classnum@
\def\getmathch@#1.#2\getmathch@{\classnum@#1 \divide\classnum@4096
 \ifcase\number\classnum@\or\or\gdef\thedots@{\dotsb@}\or
 \gdef\thedots@{\dotsb@}\fi}
{\uccode`4=`b \uccode`5=`i \uccode`6=`n
 \uppercase{\gdef\mathbin@#1#2#3{\relaxnext@
  \def\nextii@##1\mathbin@{\ifx\@sptoken\@let@token\gtest@true\fi}%
  \gtest@false\DN@##1\mathbin@{}%
 \ifx 4#1\ifx 5#2\ifx 6#3\DN@{\FN@\nextii@}\fi\fi\fi\next@}}}
{\uccode`4=`r \uccode`5=`e \uccode`6=`l
 \uppercase{\gdef\mathrel@#1#2#3{\relaxnext@
  \def\nextii@##1\mathrel@{\ifx\@sptoken\@let@token\gtest@true\fi}%
 \gtest@false\DN@##1\mathrel@{}%
 \ifx 4#1\ifx 5#2\ifx 6#3\DN@{\FN@\nextii@}\fi\fi\fi\next@}}}
{\uccode`5=`m \uccode`6=`a \uccode`7=`c
 \uppercase{\gdef\macro@#1#2#3#4\macro@{\gtest@false
  \ifx 5#1\ifx 6#2\ifx 7#3\gtest@true
  \xdef\meaning@{\macro@@#4\macro@@}\fi\fi\fi}}}
\def\macro@@#1->#2\macro@@{#2}
\newcount\DOTSCASE@
{\uccode`6=`\\ \uccode`7=`D \uccode`8=`O \uccode`9=`T \uccode`0=`S
 \uppercase{\gdef\DOTS@#1#2#3#4#5{\gtest@false\DN@##1\DOTS@{}%
  \ifx 6#1\ifx 7#2\ifx 8#3\ifx 9#4\ifx 0#5\let\next@\DOTS@@
  \fi\fi\fi\fi\fi
  \next@}}}
{\uccode`3=`B \uccode`4=`I \uccode`5=`X
 \uppercase{\gdef\DOTS@@#1{\relaxnext@
  \def\nextii@##1\DOTS@{\ifx\@sptoken\@let@token\gtest@true\fi}%
  \DN@{\FN@\nextii@}%
  \ifx 3#1\global\DOTSCASE@\z@\else
  \ifx 4#1\global\DOTSCASE@\@ne\else
  \ifx 5#1\global\DOTSCASE@\tw@\else\DN@##1\DOTS@{}%
  \fi\fi\fi\next@}}}
{\uccode`5=`\\ \uccode`6=`n \uccode`7=`o \uccode`8=`t
 \uppercase{\gdef\not@#1#2#3#4{\relaxnext@
  \def\nextii@##1\not@{\ifx\@sptoken\@let@token\gtest@true\fi}%
 \gtest@false\DN@##1\not@{}%
 \ifx 5#1\ifx 6#2\ifx 7#3\ifx 8#4\DN@{\FN@\nextii@}\fi\fi\fi
 \fi\next@}}}
\def\keybin@{\gtest@true
 \ifx\@let@token+\else\ifx\@let@token=\else
 \ifx\@let@token<\else\ifx\@let@token>\else
 \ifx\@let@token-\else\ifx\@let@token*\else\ifx\@let@token:\else
   \gtest@false\fi\fi\fi\fi\fi\fi\fi}
```

Patch to ensure `\@ldots` is defined. (Name changed to `\mathellipsis` in Dec 94 release of LaTeX.)

```
\@ifundefined{@ldots}{\def\@ldots{\mathellipsis}}{}
```

Reiterate the standard definition of `\ldots` to keep it from being clobbered by the redefinition of `\dots`.

```
\DeclareRobustCommand{\ldots}{%
  \ifmmode \mathellipsis \else \textellipsis \fi
}
\DeclareRobustCommand{\dots}{%
  \ifmmode \@xp\mdots@\else \@xp\textellipsis \fi
}
\def\tdots@{\leavevmode\unskip\relaxnext@
 \DN@{$\m@th\@ldots\,
   \ifx\@let@token,\,$\else\ifx\@let@token.\,$\else
   \ifx\@let@token;\,$\else\ifx\@let@token:\,$\else
   \ifx\@let@token?\,$\else\ifx\@let@token!\,$\else
     $ \fi\fi\fi\fi\fi\fi}%
  \ \FN@\next@}
\def\mdots@{\FN@\mdots@@}
\def\mdots@@{\gdef\thedots@{\dotso@}%
 \ifx\@let@token\boldsymbol \gdef\thedots@\boldsymbol{\boldsymboldots@}%
 \else\ifx,\@let@token \gdef\thedots@{\dotsc}%
 \else\ifx\not\@let@token \gdef\thedots@{\dotsb@}%
 \else\keybin@
 \ifgtest@\gdef\thedots@{\dotsb@}%
 \else\xdef\meaning@{\meaning\@let@token.........}%
   \xdef\meaning@@{\meaning@}%
  \@xp\math@\meaning@\math@
  \ifgtest@
   \@xp\mathch@\meaning@\mathch@
   \ifgtest@\@xp\getmathch@\meaning@\getmathch@\fi
  \else\@xp\macro@\meaning@@\macro@
  \ifgtest@
   \@xp\not@\meaning@\not@\ifgtest@\gdef\thedots@{\dotsb@}%
  \else\@xp\DOTS@\meaning@\DOTS@
  \ifgtest@
   \ifcase\number\DOTSCASE@\gdef\thedots@{\dotsb@}%
    \or\gdef\thedots@{\dotsi}\else\fi
  \else\@xp\math@\meaning@\math@
  \ifgtest@\@xp\mathbin@\meaning@\mathbin@
  \ifgtest@\gdef\thedots@{\dotsb@}%
  \else\@xp\mathrel@\meaning@\mathrel@
  \ifgtest@\gdef\thedots@{\dotsb@}%
  \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi
 \thedots@}
```

The = character is necessary in the two `\let` assignments in `\boldsymboldots@`, because the symbol we are making bold might be an = sign.

```
\def\boldsymboldots@#1{%
  \bold@true\let\@let@token=#1\let\delayed@=#1\mdots@@
  \boldsymbol#1\bold@false}
```

The definition of `\@cdots` is merely the `plain.tex` definition of `\cdots`.

```
\ams@def\@cdots{\mathinner{\cdotp\cdotp\cdotp}}
\newcommand{\dotsi}{\!\@cdots}
\let\dotsb@\@cdots
```

If any new right delimiters are defined, they would need to be added to the definition of `\rightdelim@` in order for `\dots` to work properly in all cases.

```
\def\rightdelim@{\gtest@true
 \ifx\@let@token)\else
 \ifx\@let@token]\else
 \ifx\@let@token\rbrack\else
```

```
 \ifx\@let@token\}\else
 \ifx\@let@token\rbrace\else
 \ifx\@let@token\rangle\else
 \ifx\@let@token\rceil\else
 \ifx\@let@token\rfloor\else
 \ifx\@let@token\rgroup\else
 \ifx\@let@token\rmoustache\else
 \ifx\@let@token\right\else
 \ifx\@let@token\bigr\else
 \ifx\@let@token\biggr\else
 \ifx\@let@token\Bigr\else
 \ifx\@let@token\Biggr\else\gtest@false
 \fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi\fi}
\def\extra@{%
 \rightdelim@\ifgtest@
 \else\ifx\@let@token$\gtest@true
 \else\xdef\meaning@{\meaning\@let@token.........}%
 \@xp\macro@\meaning@\macro@\ifgtest@
 \@xp\DOTS@\meaning@\DOTS@
 \ifgtest@
 \ifnum\DOTSCASE@=\tw@\gtest@true\else\gtest@false
 \fi\fi\fi\fi\fi}
\newif\ifbold@
\def\dotso@{\relaxnext@
 \ifbold@
  \let\@let@token\delayed@
  \def\nextii@{\extra@\@ldots\ifgtest@\,\fi}%
 \else
  \def\nextii@{\DN@{\extra@\@ldots\ifgtest@\,\fi}\FN@\next@}%
 \fi
 \nextii@}
```

Why not save some tokens? (space vs. time).

```
\def\extrap@#1{%
 \DN@{#1\,}%
 \ifx\@let@token,\else
 \ifx\@let@token;\else
 \ifx\@let@token.\else\extra@
 \ifgtest@\else
 \let\next@#1\fi\fi\fi\fi\next@}
```

The \cdots command.

```
\ams@DeclareRobustCommand{\cdots}{\DN@{\extrap@\@cdots}\FN@\next@}
\let\dotsb\cdots
\let\dotsm\cdots
\DeclareRobustCommand{\dotso}{\relax
  \ifmmode \DN@{\extrap@\@ldots}%
  \else \let\next@\tdots@\fi
  \FN@\next@}
\DeclareRobustCommand{\dotsc}{%
  \DN@{\ifx\@let@token;\@ldots\,%
       \else \ifx\@let@token.\@ldots\,%
       \else \extra@\@ldots \ifgtest@\,\fi
       \fi\fi}%
  \FN@\next@}
\renewcommand{\longrightarrow}{%
  \DOTSB\protect\relbar\protect\joinrel\rightarrow}
\renewcommand{\Longrightarrow}{%
  \DOTSB\protect\Relbar\protect\joinrel\Rightarrow}
\renewcommand{\longleftarrow}{%
```

```
  \DOTSB\leftarrow\protect\joinrel\protect\relbar}
\renewcommand{\Longleftarrow}{%
  \DOTSB\Leftarrow\protect\joinrel\protect\Relbar}
\renewcommand{\longleftrightarrow}{\DOTSB\leftarrow\joinrel\rightarrow}
\renewcommand{\Longleftrightarrow}{\DOTSB\Leftarrow\joinrel\Rightarrow}
\renewcommand{\mapsto}{\DOTSB\mapstochar\rightarrow}
\renewcommand{\longmapsto}{\DOTSB\mapstochar\longrightarrow}
\renewcommand{\hookrightarrow}{\DOTSB\lhook\joinrel\rightarrow}
\renewcommand{\hookleftarrow}{\DOTSB\leftarrow\joinrel\rhook}
\renewcommand{\iff}{\DOTSB\;\Longleftrightarrow\;}
```

The `\doteq` command formerly used `\buildrel`; we avoid that because it requires '`\over`' as part of its syntax. Use 0pt instead of `\z@` for robustitude.

```
\renewcommand{\doteq}{%
  \DOTSB\mathrel{\mathop{\kern0pt =}\limits^{\textstyle.}}}
```

8 Integral signs

The straightforward `\ifinner` test to see if the current math context is non-display, fails if, for instance, we are typesetting a multiline display within an `\halign`, with the pieces going into constructions like

```
$\displaystyle...$
```

So we need a better test to find out if we are 'in a display'. We therefore create `\if@display`.

```
\newif\if@display
\everydisplay\@xp{\the\everydisplay \@displaytrue}
```

Default value for integral limits is `\nolimits`, see the definition of the 'nointlimits' option.

```
\renewcommand{\int}{\DOTSI\intop\ilimits@}
\renewcommand{\oint}{\DOTSI\ointop\ilimits@}
\def\intkern@{\mkern-6mu\mathchoice{\mkern-3mu}{}{}{}}
\def\intdots@{\mathchoice{\@cdots}%
 {{\cdotp}\mkern1.5mu{\cdotp}\mkern1.5mu{\cdotp}}%
 {{\cdotp}\mkern1mu{\cdotp}\mkern1mu{\cdotp}}%
 {{\cdotp}\mkern1mu{\cdotp}\mkern1mu{\cdotp}}}
%
\ams@newcommand{\iint}{\DOTSI\protect\MultiIntegral{2}}
\ams@newcommand{\iiint}{\DOTSI\protect\MultiIntegral{3}}
\ams@newcommand{\iiiint}{\DOTSI\protect\MultiIntegral{4}}
\newcommand{\idotsint}{\DOTSI\protect\MultiIntegral{0}}
```

If the `\limits` option is applied, use `\mathop` and fudge the left-hand space a bit to make the subscript visually centered.

`#1` is the multiplicity.

```
\newcommand{\MultiIntegral}[1]{%
  \edef\ints@c{\noexpand\intop
    \ifnum#1=\z@\noexpand\intdots@\else\noexpand\intkern@\fi
    \ifnum#1>\tw@\noexpand\intop\noexpand\intkern@\fi
    \ifnum#1>\thr@@\noexpand\intop\noexpand\intkern@\fi
    \noexpand\intop
    \noexpand\ilimits@
  }%
  \futurelet\@let@token\ints@a
}

\def\ints@a{%
  \ifx\limits\@let@token \ints@b
```

```
  \else \ifx\displaylimits\@let@token \ints@b
  \else\ifx\ilimits@\displaylimits \ints@b
  \fi\fi\fi
  \ints@c
}
\def\ints@b{%
  \mkern-7mu\mathchoice{\mkern-2mu}{}{}{}%
  \mathop\bgroup
    \mkern7mu\mathchoice{\mkern2mu}{}{}{}%
    \let\ilimits@\egroup
}%
```

9 Size dependent definitions

We now define all stuff which has to change whenever a new math size is to be activated. LaTeX provides a hook called `\every@math@size` to support such a need. All assignments in the `\every@math@size` hook that need to take outside effect should be global.

9.1 Struts for math

The various kinds of struts could use some analysis and perhaps consolidation.

For example perhaps the `\bBigg` delimiters could use

```
1.2\ht\strutbox (1.8, 2.4, 3.0)
```

instead of

```
1.0\big@size (1.5, 2.0, 2.5)
```

since `\strut` is reset with every size change [mjd, 1994/10/07]. But this change would introduce the possibility of changed line and page breaks in existing documents, so would need to be handled with care.

`\Mathstrut@` `\Mathstrutbox@` `\resetMathstrut@`

Here comes the code for Spivak's `\Mathstrut@`.

```
\newbox\Mathstrutbox@
\setbox\Mathstrutbox@=\hbox{}
\def\Mathstrut@{\copy\Mathstrutbox@}
```

The setting of the height and depth of the `\Mathstrutbox@` is done in the `\every@math@size` hook since it depends on the height of a paren. As `\every@math@size` is triggered by `$` after a font size change, we want to avoid using another math formula `$...$` to measure the math paren height; instead we go through the mathcode of the `(` character. We assume that the mathcode has a leading hex digit 4 indicating 'open delimiter'; this allows us to make a relatively simple function to get the correct font and character position.

```
\def\resetMathstrut@{%
  \setbox\z@\hbox{%
    \mathchardef\@tempa\mathcode`\(\relax
    \def\@tempb##1"##2##3{\the\textfont"##3\char"}%
    \expandafter\@tempb\meaning\@tempa \relax
  }%
```

These height and depth assignments are implicitly global.

```
  \ht\Mathstrutbox@\ht\z@ \dp\Mathstrutbox@\dp\z@
}
\addto@hook\every@math@size{\resetMathstrut@}
```

`\strut@` `\strutbox@`

Next follows a special internal strut which is supposed to match the height and the depth of a normal `\strut` minus `\normallineskiplimit` according to M. Spivak.

This should really go into the definition of \size@update, and then the box reset could be local; but \size@update doesn't have any hook and is handled in such a way that it cannot even be changed except by changing \set@fontsize. So instead we put \reset@strutbox@ into \every@math@size and make it global. Then because of some complications in the way \glb@settings and \check@mathfonts work, we have to re-invoke it at the beginning of every environment that might use \strut@. Fortunately this can be achieved (more or less) through the \spread@equation hook. [mjd,2000/03/10]

```
\newbox\strutbox@
\def\strut@{\copy\strutbox@}
\def\reset@strutbox@{%
  \global\setbox\strutbox@\hbox{%
    \lower.5\normallineskiplimit
       \vbox{\kern-\normallineskiplimit\copy\strutbox}}}
\addto@hook\every@math@size{\reset@strutbox@}
\AtBeginDocument{\reset@strutbox@}
```

9.2 Big delimiters

We are now going to redefine the plain TEX commands \big, \bigl, etc., to produce different results in different sizes. Actually we only have to define \big, \Big, etc., since they are used to construct the directional versions \bigl, \bigr, and the rest.

\big \Big \bigg \Bigg

To save token space we put everything into the common macro \bBigg@. The macros are now simply a call to \bBigg@ with a factor to determine the correct height of the delimiter as an argument. This code should better go into a future version of the LATEX kernel; the macro \n@space is then superfluous (since it is only used once) and should be removed to avoid wasting hash table space unnecessarily.

```
\renewcommand{\big}{\bBigg@\@ne}
\renewcommand{\Big}{\bBigg@{1.5}}
\renewcommand{\bigg}{\bBigg@\tw@}
\renewcommand{\Bigg}{\bBigg@{2.5}}
```

\bBigg@

Now we tackle the macro which has to do the real work. It actually has two arguments, the factor and the wanted delimiter.

```
\def\bBigg@#1#2{%
```

We start with an extra set of braces because we want constructions like \def\bigl{\mathopen\big} to work without the overhead of extra arguments.

```
  {\@mathmeasure\z@{\nulldelimiterspace\z@}%
     {\left#2\vcenter to#1\big@size{}\right.}%
   \box\z@}}
```

\big@size

\big@size needs to be set to 1.2 times the height of a math paren. This height is already recorded in \Mathstrutbox@.

```
\addto@hook\every@math@size{%
  \global\big@size 1.2\ht\Mathstrutbox@
  \global\advance\big@size 1.2\dp\Mathstrutbox@ }
\newdimen\big@size
```

10 Math accents

We want to change the leading digit of math accents to be \accentclass@ so that it can vary according to certain internal purposes.

```
\def\accentclass@{7}
\def\noaccents@{\def\accentclass@{0}}
```

There are a few ⟨*math alphabet*⟩s in the standard fonts where we have to change the extra macros because the standard definitions don't account for these accent problems. The first is for the `\mathit` command.

```
\DeclareFontEncoding{OML}{}{\noaccents@}
```

The next one corrects the `\cal` alphabet.

```
\DeclareFontEncoding{OMS}{}{\noaccents@}
```

Triple and quadruple dot accents.

```
\ams@newcommand{\dddot}[1]{%
  {\mathop{#1}\limits^{\vbox to-1.4\ex@{\kern-\tw@\ex@
   \hbox{\normalfont ...}\vss}}}}
\ams@newcommand{\ddddot}[1]{%
  {\mathop{#1}\limits^{\vbox to-1.4\ex@{\kern-\tw@\ex@
   \hbox{\normalfont....}\vss}}}}
```

The following code deals with support for compound accents. By redefining `\set@mathaccent` we ensure that `\DeclareMathAccent` will define accent commands to run our `\mathaccentV` function instead of the primitive `\mathaccent`.

```
\def\set@mathaccent#1#2#3#4{%
  \xdef#2{\@nx\protect\@nx\mathaccentV
    {\@xp\@gobble\string#2}\hexnumber@#1#4}%
}
```

We redefine the standard math accent commands to call `\mathaccentV`, using the mathgroup/encoding-number information embedded in their previous definitions. If the definition of an accent command does not have the expected form, we leave the accent command alone, but give a warning. For widehat and widetilde, we need to avoid clobbering the definitions done by the `amsfonts` package. Arbitrating the contention between `amsmath` and `amsfonts` to allow doubling a widetilde accent looks tricky, so for the time being [mjd,1999/07/19] we just leave `\widehat` and `\widetilde` alone. As a result, if the `amsmath` package is loaded on top of a vanilla LaTeX documentclass, everything runs through with no warnings. If a Lucida Math or other math fonts package is loaded in addition to `amsmath`, there are greater difficulties, but those are addressed elsewhere.

```
\def\@tempa#1{\@xp\@tempb\meaning#1\@nil#1}
\def\@tempb#1>#2#3 #4\@nil#5{%
  \@xp\ifx\csname#3\endcsname\mathaccent
    \@tempc#4?"7777\@nil#5%
  \else
    \PackageWarningNoLine{amsmath}{%
      Unable to redefine math accent \string#5}%
  \fi
}
\def\@tempc#1"#2#3#4#5#6\@nil#7{%
  \chardef\@tempd="#3\relax\set@mathaccent\@tempd{#7}{#2}{#4#5}}
\@tempa{\hat}
\@tempa{\check}
\@tempa{\tilde}
\@tempa{\acute}
\@tempa{\grave}
\@tempa{\dot}
\@tempa{\ddot}
\@tempa{\breve}
\@tempa{\bar}
\@tempa{\vec}
\@ifundefined{mathring}{%
  \DeclareMathAccent{\mathring}{\mathalpha}{operators}{"17}
```

```
}{%
  \@tempa{\mathring}
}
%%\@tempa\widetilde
%%\@tempa\widehat
```

Regression testing of amsmath 2.0 showed that in some documents there occurred fragments of the form

```
\hat\mathcal{G}
```

This is not at all correct syntax for the argument of a LaTeX command but it produced the intended result anyway because of the internal syntax of the `\mathaccent` primitive. With `\mathaccentV`, it will yield an error message. We therefore do a special check for such syntax problems in order to make the error message more informative. (dmj: ??????)

```
\newcommand{\acc@check}{}
\newcommand{\acc@error}{}
\def\acc@check{\@ifnextchar\@empty\relax\acc@error}
```

We put most of the tokens in a separate macro so they do not get scanned unless they are actually needed.

```
\def\acc@error{%
  \@amsmath@err{%
    Improper argument for math accent:\MessageBreak
    Extra braces must be added to prevent wrong output%
  }\@ehc
}
```

For `\mathaccentV` part of the processing is dependent on the depth of nesting of math accent commands. We introduce a dedicated counter for this instead of using chardef because we want to increment/decrement it during processing, and incrementing a chardef integer is more work.

```
\newcount\macc@depth
```

Provide this function in case it is not already available.

```
\long\def\@gobblethree#1#2#3{}
```

The `\mathaccentV` function first counts the number of nested math accents by setting the argument in a throw-away box. (This is not as risky as such an operation would normally be because the argument is generally either a simple math symbol or a nested math accent call with a simple math symbol at the bottom of the nesting.)

There are two benefits from counting the nesting levels first before doing anything else: (1) we can fall back to a simple `\mathaccent` call if the nesting depth is 1, and (2) if the nesting depth is greater than 1, we would like to be able to tell when we have reached the lowest level, because at that point we want to save the argument for later use and place an accent on top of a phantom copy.

When we have multiple accents, they will be placed on top of the invisible box, followed by some suitable kerns, then a visible copy of the nucleus. To see why, let us look at what goes wrong with a double application of the `\mathaccent` primitive. The standard definition of `\hat` is `\mathaccent"705E`, so `\hat{\hat{F}}` expands to

```
\mathaccent"705E{\mathaccent"705E{F}}
```

The result of this operation is

```
\vbox(12.11111+0.0)x7.81946
.\hbox(6.94444+0.0)x0.0, shifted 1.40973
..\OT1/cmr/m/n/10 ^
.\kern-4.30554
```

```
.\vbox(9.47221+0.0)x7.81946
..\hbox(6.94444+0.0)x0.0, shifted 2.24309
...\OT1/cmr/m/n/10 ^
..\kern-4.30554
..\hbox(6.83331+0.0)x7.81946
...\OML/cmm/m/it/10 F
```

TEX starts by constructing a vbox with the hat character on top of the F. Then it puts another hat character on top of the vbox; but without skew information, because that is only applied by `\mathaccent` when the base object is a simple symbol. So the first accent is skewed to the correct position but all later accents are not. By the way, the actual width of the F in the above example is less than 7.81946; the box in which it is packed was automatically lengthened by the width of the F's italic correction (without actually putting in a kern for it).

To get the second accent shifted farther to the right we artificially increase the width of the innermost box and add a compensating kern afterward. Furthermore, to get proper placement of a following subscript or superscript, we take the base symbol out, leaving a phantom in its place, and print it by itself following the kern. We then need to increase the kern amount to move the base character backward under the accents again. Here is what the results look like:

```
\vbox(12.11111+0.0)x9.48618
.\hbox(6.94444+0.0)x0.0, shifted 2.24309
..\OT1/cmr/m/n/10 ^
.\kern-4.30554
.\vbox(9.47221+0.0)x9.48618
..\hbox(6.94444+0.0)x0.0, shifted 2.24309
...\OT1/cmr/m/n/10 ^
..\kern-4.30554
..\hbox(6.83331+0.0)x9.48618
...\hbox(6.83331+0.0)x7.81946
...\kern 1.66672
\kern -9.48618
\OML/cmm/m/it/10 F
```

Much of this implementation is based on code from the `accents` package of Javier Bezos. I added the test to revert to a simple `\mathaccent` when accents are not nested, and some other refinements to reduce the number of kerns used (to conserve box memory) and the number of cycles through `\mathchoice` (to make things run a little faster). It was all rather difficult and my first two attempts had serious bugs but I hope and believe that this version will do better. [mjd,2000/03/15]

The "V" in `\mathaccentV` is just an indication that it takes five arguments. It is important that the name includes `mathaccent`, otherwise `\DeclareMathAccent` will balk at redefining one of our accent commands, for example when an alternative math font package is loaded.

```
\def\mathaccentV#1#2#3#4#5{%
  \ifmmode
    \gdef\macc@tmp{\macc@depth\@ne}%
    \setbox\z@\hbox{%
      \let\mathaccentV\macc@test
      \let\use@mathgroup\@gobbletwo \let\select@group\@gobblethree
      \frozen@everymath{}$#5$%
    }%
    \macc@tmp
    \ifnum\macc@depth=\@ne
      \global\let\macc@nucleus\@empty
      \mathaccent"\accentclass@
    \else
```

```
      \@xp\macc@nested
    \fi
    #2#3#4{#5}%
    \macc@nucleus
  \else
    \@xp\nonmatherr@\csname#1\endcsname
  \fi
}
\def\macc@test#1#2#3#4{\xdef\macc@tmp{\macc@tmp\advance\macc@depth\@ne}}
\def\macc@group{-1}
\def\macc@nested#1#2#3#4{%
  \begingroup
  \let\math@bgroup\@empty \let\math@egroup\macc@set@skewchar
  \mathsurround\z@ \frozen@everymath{\mathgroup\macc@group\relax}%
  \macc@set@skewchar\relax
  \let\mathaccentV\macc@nested@a
  \macc@nested@a\relax#1#2#3{#4}%
  \endgroup
}
\let\macc@palette\mathpalette
\def\macc@nested@a#1#2#3#4#5{%
```

This test saves some work that would otherwise be always repeated fourfold thanks to `\mathchoice`.

```
  \ifnum\macc@group=\mathgroup
  \else \macc@set@skewchar\relax \edef\macc@group{\the\mathgroup}%
  \fi
  \mathchardef\macc@code "\accentclass@ #2#3#4\relax
  \macc@palette\macc@a{#5}%
}
```

The reason that `\macc@set@skewchar` takes an argument is so that it can serve as a direct substitute for `\math@egroup`, in addition to being used separately.

Setting a skewchar with this method works for symbols of variable mathgroup (class 7, letters and numbers) but not necessarily for special symbols like `\partial` or `\xi` whose mathgroup doesn't change; fortunately the most commonly used ones come from mathgroup one, which is the fall-back mathgroup for skewchar.

```
\def\macc@set@skewchar#1{%
  \begingroup
  \ifnum\mathgroup=\m@ne \let\@tempa\@ne
  \else
    \ifnum\skewchar\textfont\mathgroup=\m@ne \let\@tempa\@ne
    \else \let\@tempa\mathgroup
    \fi
  \fi
  \count@=\skewchar\textfont\@tempa
  \advance\count@"7100
  \edef\@tempa{\endgroup
    \mathchardef\noexpand\macc@skewchar=\number\count@\relax}%
  \@tempa
  #1%
}
```

Arg1 is math-style, arg2 is accent base object. We assume that math style doesn't change within the nested group of accents; this means we can set `\macc@style` only once and redefine `\macc@palette` to use it, in order to run

\mathchoice only once instead of multiplying the calls exponentially as the nesting level increases.

```
\def\macc@a#1#2{%
  \begingroup
  \let\macc@style#1\relax
  \def\macc@palette##1{##1\macc@style}%
  \advance\macc@depth\m@ne
  \ifnum\macc@depth=\z@
    \gdef\macc@nucleus{#2}%
```

Extra \@empty tokens are to prevent low-level TEX errors from the potential syntactic error that \acc@check checks for.

```
    \setbox\z@\hbox{$#1#2\@empty{}\macc@skewchar$}%
    \setbox\tw@\hbox{$#1#2\@empty\macc@skewchar$}%
    \dimen@\tw@\wd\tw@ \advance\dimen@-\tw@\wd\z@
    \xdef\macc@kerna{\the\dimen@\relax}%
    \setbox4\hbox{$#1#2\acc@check\@empty$}%
    \global\setbox\@ne\hbox to\wd4{}%
    \ht\@ne\ht4 \dp\@ne\dp4
    \xdef\macc@kernb{\the\wd4\relax}%
    \mathaccent\macc@code{\box\@ne\kern\macc@kerna}%
  \else
    \mathaccent\macc@code{\let\macc@adjust\@empty #1#2\@empty}%
    \macc@adjust
  \fi
  \endgroup
}
```

```
\def\macc@adjust{%
  \dimen@\macc@kerna\advance\dimen@\macc@kernb
  \kern-\dimen@
}
```

The commands \Hat, \Tilde, …, are supported as synonyms of \hat, \tilde, …, for backward compatibility.

```
\def\Hat{\hat}
\def\Check{\check}
\def\Tilde{\tilde}
\def\Acute{\acute}
\def\Grave{\grave}
\def\Dot{\dot}
\def\Ddot{\ddot}
\def\Breve{\breve}
\def\Bar{\bar}
\def\Vec{\vec}
```

This error message about math mode is used several times so we make an abbreviation for it.

```
\def\nonmatherr@#1{\@amsmath@err{\protect
  #1 allowed only in math mode}\@ehd}
```

11 Mods, continued fractions, etc.

The commands \bmod, \pmod, \pod, \mod aren't currently robust. [mjd, 1994/09/05]

```
\renewcommand{\bmod}{\nonscript\mskip-\medmuskip\mkern5mu\mathbin
  {\operator@font mod}\penalty900
  \mkern5mu\nonscript\mskip-\medmuskip}
\newcommand{\pod}[1]{\allowbreak
  \if@display\mkern18mu\else\mkern8mu\fi(#1)}
```

```
\renewcommand{\pmod}[1]{\pod{{\operator@font mod}\mkern6mu#1}}
\newcommand{\mod}[1]{\allowbreak\if@display\mkern18mu
  \else\mkern12mu\fi{\operator@font mod}\,\,#1}
```

Continued fractions. The optional arg l or r controls horizontal placement of the numerators. The `\kern-\nulldelimiterspace` is needed in the definition if we want the right-hand sides of the fraction rules to line up. The `\strut` keeps the numerator of a subsidiary cfrac from coming too close to the fraction rule above it.

```
\newcommand{\cfrac}[3][c]{{\displaystyle\frac{%
  \strut\ifx r#1\hfill\fi#2\ifx l#1\hfill\fi}{#3}}%
  \kern-\nulldelimiterspace}
```

`\overset` and `\underset` put symbols above, respectively below, a symbol that is not a `\mathop` and therefore does not naturally accept limits. `\binrel@@` uses information collected by `\binrel@` to make the resulting construction be of type mathrel or mathbin if the base symbol is either of those types.

```
\newcommand{\overset}[2]{\binrel@{#2}%
  \binrel@@{\mathop{\kern\z@#2}\limits^{#1}}}
\newcommand{\underset}[2]{\binrel@{#2}%
  \binrel@@{\mathop{\kern\z@#2}\limits_{#1}}}
```

`\sideset` allows placing 'adscript' symbols at the four corners of a `\mathop`, *in addition to* limits. Left-side adscripts go into arg `#1`, in the form `_{...}^{...}`, and right-side adscripts go into arg `#2`.

As currently written [mjd, 1995/01/21] this is pretty haphazard. In order to really make it work properly in full generality we'd have to read and measure the top and bottom limits and use mathchoice to always get the right mathstyle for each piece, etc., etc.

```
\newcommand{\sideset}[3]{%
  \@mathmeasure\z@\displaystyle{#3}%
```

Use a global box assignment here since the depth override is implicitly global. Then move the constructed box to a local box register (2) to ensure it won't get destroyed during the next two mathmeasure statements. This precaution may be more extreme than necessary in practice.

```
  \global\setbox\@ne\vbox to\ht\z@{}\dp\@ne\dp\z@
  \setbox\tw@\box\@ne
  \@mathmeasure4\displaystyle{\copy\tw@#1}%
  \@mathmeasure6\displaystyle{#3\nolimits#2}%
  \dimen@-\wd6 \advance\dimen@\wd4 \advance\dimen@\wd\z@
  \hbox to\dimen@{}\mathop{\kern-\dimen@\box4\box6}%
}
```

`\smash` We add to the `\smash` command an optional argument denoting the part of the formula to be smashed.

```
\renewcommand{\smash}[1][tb]{%
  \def\mb@t{\ht}\def\mb@b{\dp}\def\mb@tb{\ht\z@\z@\dp}%
  \edef\finsm@sh{\csname mb@#1\endcsname\z@\z@ \box\z@}%
  \ifmmode \@xp\mathpalette\@xp\mathsm@sh
  \else \@xp\makesm@sh
  \fi
}
```

12 Extensible arrows

The minus sign used in constructing these arrow fills is smashed so that superscripts above the arrows won't be too high. This primarily affects the `\xleftarrow` and `\xrightarrow` arrows.

```
\mathchardef\std@minus\mathcode`\-\relax
\mathchardef\std@equal\mathcode`\=\relax
```

In case some alternative math fonts are loaded later:

```
\AtBeginDocument{%
  \mathchardef\std@minus\mathcode`\-\relax
  \mathchardef\std@equal\mathcode`\=\relax
}

\ams@def\relbar{\mathrel{\mathpalette\mathsm@sh\std@minus}}
\ams@def\Relbar{\mathrel\std@equal}

\def\arrowfill@#1#2#3#4{%
  $\m@th\thickmuskip0mu\medmuskip\thickmuskip\thinmuskip\thickmuskip
   \relax#4#1\mkern-7mu%
   \cleaders\hbox{$#4\mkern-2mu#2\mkern-2mu$}\hfill
   \mkern-7mu#3$%
}
\def\leftarrowfill@{\arrowfill@\leftarrow\relbar\relbar}
\def\rightarrowfill@{\arrowfill@\relbar\relbar\rightarrow}
\def\leftrightarrowfill@{\arrowfill@\leftarrow\relbar\rightarrow}
\def\Leftarrowfill@{\arrowfill@\Leftarrow\Relbar\Relbar}
\def\Rightarrowfill@{\arrowfill@\Relbar\Relbar\Rightarrow}
\def\Leftrightarrowfill@{\arrowfill@\Leftarrow\Relbar\Rightarrow}

\def\overarrow@#1#2#3{\vbox{\ialign{##\crcr#1#2\crcr
 \noalign{\nointerlineskip}$\m@th\hfil#2#3\hfil$\crcr}}}
\ams@renewcommand{\overrightarrow}{%
  \mathpalette{\overarrow@\rightarrowfill@}}
\ams@renewcommand{\overleftarrow}{%
  \mathpalette{\overarrow@\leftarrowfill@}}
\ams@newcommand{\overleftrightarrow}{%
  \mathpalette{\overarrow@\leftrightarrowfill@}}

\def\underarrow@#1#2#3{%
 \vtop{\ialign{##\crcr$\m@th\hfil#2#3\hfil$\crcr
 \noalign{\nointerlineskip\kern1.3\ex@}#1#2\crcr}}}
\ams@newcommand{\underrightarrow}{%
  \mathpalette{\underarrow@\rightarrowfill@}}
\ams@newcommand{\underleftarrow}{%
  \mathpalette{\underarrow@\leftarrowfill@}}
\ams@newcommand{\underleftrightarrow}{%
  \mathpalette{\underarrow@\leftrightarrowfill@}}

%\newcommand{\xrightarrow}[2][]{\ext@arrow 0359\rightarrowfill@{#1}{#2}}
\def\ext@arrow#1#2#3#4#5#6#7{%
  \mathrel{\mathop{%
```

Measure the superscript and subscript.

```
    \setbox\z@\hbox{#5\displaystyle}%
    \setbox\tw@\vbox{\m@th
      \hbox{$\scriptstyle\mkern#3mu{#6}\mkern#4mu$}%
      \hbox{$\scriptstyle\mkern#3mu{#7}\mkern#4mu$}%
      \copy\z@
    }%
    \hbox to\wd\tw@{\unhbox\z@}}%
```

We don't want to place an empty subscript since that will produce too much blank space below the arrow.

```
  \limits
    \@ifnotempty{#7}{^{\if0#1\else\mkern#1mu\fi
                       #7\if0#2\else\mkern#2mu\fi}}%
    \@ifnotempty{#6}{_{\if0#1\else\mkern#1mu\fi
```

```
                        #6\if0#2\else\mkern#2mu\fi}}}%
}
```

Some extensible arrows to serve as mathrels and taking sub/superscripts. These commands are robust because they take an optional argument.

```
\newcommand{\xrightarrow}[2][]{\ext@arrow 0359\rightarrowfill@{#1}{#2}}
\newcommand{\xleftarrow}[2][]{\ext@arrow 3095\leftarrowfill@{#1}{#2}}
```

13 Array-related environments

13.1 Remarks

Because these environments can be nested within the equation structures that allow `\tag`, there is some cross-influence in the internal workings of the `\\` command.

13.2 The `\substack` command

The `\substack` command can be used to set subscripts and superscripts that consist of several lines. Usage:

```
X_{\substack{a=1\\b=2}}
```

`subarray` The `subarray` environment makes a small-size array suitable for use in a subscript or superscript. At the moment the supported arguments are not the full possibilities of `array` but only `c` or `l` for centered or left-aligned. And only one column.

```
\newenvironment{subarray}[1]{%
```

Note: The predecessors of `subarray` (`Sb` and `Sp`, inherited from $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX) used `\vbox` instead of `\vcenter`. But when a multiline subscript is placed in `\limits` position `\vcenter` is no worse than `\vbox`, and when it is placed in the `\nolimits` position (e.g., for an integral), `\vcenter` provides clearly better positioning than `\vbox`.

```
  \vcenter\bgroup
```

Use `\Let@` to set the proper meaning of the `\\` and `\\*` commands. And restore the meaning of `\math@cr@@@` to `\cr` (see above) in case `subarray` is used inside one of the more complicated alignment macros where the meaning of `\math@cr@@@` is different. Similarly, call `\default@tag` to ensure that a line break here doesn't get an equation number!

```
  \Let@ \restore@math@cr \default@tag
```

Set the line spacing to be the same as `\atop` (when `\atop` occurs in `\textstyle` or smaller), cf *The TEXbook*, Appendix G.

```
  \baselineskip\fontdimen10 \scriptfont\tw@
  \advance\baselineskip\fontdimen12 \scriptfont\tw@
  \lineskip\thr@@\fontdimen8 \scriptfont\thr@@
  \lineskiplimit\lineskip
```

Start the `\vbox \halign` structure that encloses the contents. Notice that we never get `\scriptscriptstyle`. That would require a `\mathchoice` (ugh).

```
  \ialign\bgroup\ifx c#1\hfil\fi
    $\m@th\scriptstyle##$\hfil\crcr
}{%
  \crcr\egroup\egroup
}
```

`\substack` The `\substack` command is just an abbreviation for the most common use of `subarray`.

```
\newcommand{\substack}[1]{\subarray{c}#1\endsubarray}
```

13.3 Matrices

smallmatrix

`smallmatrix` is again an alignment, this time in a centered box. The opening incantations are basically the same as those in `\multilimits@`, followed by the alignment itself. A remark: the baselineskip (`9\ex@`) used in $\mathcal{AMS}$-TEX is too large for use in text with the usual baselineskip of 12 or 13 points; we change it here to `6\ex@` and also adjust the `\lineskip` and `\lineskiplimit` slightly to compensate. (MJD)

```
\newenvironment{smallmatrix}{\null\,\vcenter\bgroup
  \Let@\restore@math@cr\default@tag
  \baselineskip6\ex@ \lineskip1.5\ex@ \lineskiplimit\lineskip
  \ialign\bgroup\hfil$\m@th\scriptstyle##$\hfil&&\thickspace\hfil
  $\m@th\scriptstyle##$\hfil\crcr
}{%
  \crcr\egroup\egroup\,%
}
```

matrix

The `matrix` environment is just an `array` that provides up to ten centered columns, so that users don't have to give the col-spec argument explicitly—unless they want some of the columns noncentered, that is. The maximum number of columns is actually not fixed at ten but given by the counter `MatrixCols`, and can therefore be increased by changing that counter.

The extra space of `\arraycolsep` that `array` adds on each side is a waste so we remove it here (perhaps we should instead remove it from `array` in general, but that's a harder task).

TODO: Think about re-implementing `\matrix` to get rid of the `\c@MatrixCols` limit and have hard-wired preamble that doesn't have to be rebuilt each time.

We must use `\renewenvironment` for `matrix` and `pmatrix` because LaTeX doesn't kill the definitions found in `plain.tex`, even though it probably should because of their foreign syntax.

```
\renewenvironment{matrix}{%
  \matrix@check\matrix\env@matrix
}{%
  \endarray \hskip -\arraycolsep
}
```

\env@matrix

```
\def\env@matrix{\hskip -\arraycolsep
  \let\@ifnextchar\new@ifnextchar
  \array{*\c@MaxMatrixCols c}}
```

\c@MaxMatrixCols

```
\newcount\c@MaxMatrixCols \c@MaxMatrixCols=10
```

\matrix@check

For various reasons, authors sometimes use the Plain TEX form of `\matrix` or `\pmatrix` in LaTeX documents. If they later add an invocation of the `amsmath` package to their document, the Plain TEX syntax would lead to rather unintelligible error messages. The `\matrix@check` function does some checking to forestall that problem.

```
\def\matrix@check#1{%
  \@xp\ifx\csname\@currenvir\endcsname#1%
  \else\matrix@error#1%
```

This error recovery is not that good but is better than the infinite loop that can result from calling `\array` without a matching `\endarray`. (The array setup leaves `\par` empty.)

```
    \@xp\@gobble
  \fi
```

```
}
```

\matrix@error

```
\def\matrix@error#1{%
  \@amsmath@err{%
Old form `\string#1' should be \string\begin{\@xp\@gobble\string#1}%
  }{%
`\string#1{...}' is old Plain-TeX syntax whose use is
ill-advised in LaTeX.%
  }%
}
\renewenvironment{pmatrix}{%
  \left(%
  \matrix@check\pmatrix\env@matrix
}{
  \endmatrix\right)%
}
\newenvironment{bmatrix}{\left[\env@matrix}{\endmatrix\right]}
\newenvironment{Bmatrix}{%
  \left\lbrace\env@matrix
}{%
  \endmatrix\right\rbrace
}
\newenvironment{vmatrix}{\left\lvert\env@matrix}{\endmatrix\right\rvert}
\newenvironment{Vmatrix}{\left\lVert\env@matrix}{\endmatrix\right\rVert}
\let\hdots\@ldots
\newcommand{\hdotsfor}[1]{%
  \ifx[#1\@xp\shdots@for\else\hdots@for\@ne{#1}\fi}
\newmuskip\dotsspace@
\def\shdots@for#1]{\hdots@for{#1}}
\def\hdots@for#1#2{\multicolumn{#2}c%
  {\m@th\dotsspace@1.5mu\mkern-#1\dotsspace@
   \xleaders\hbox{$\m@th\mkern#1\dotsspace@.\mkern#1\dotsspace@$}%
           \hfill
   \mkern-#1\dotsspace@}%
   }
```

cases

The easiest way to produce the `cases` environment is to base it on the `array` environment. We must use `\renewenvironment` to override the definition of `\cases` that LaTeX (unwisely) leaves in place from `plain.tex`.

```
\renewenvironment{cases}{%
  \matrix@check\cases\env@cases
}{%
  \endarray\right.%
}
\def\env@cases{%
  \let\@ifnextchar\new@ifnextchar
  \left\lbrace
  \def\arraystretch{1.2}%
  \array{@{}l@{\quad}l@{}}%
}
```

14 Equation sub-numbering

```
\newcounter{parentequation}% Counter for ``parent equation''.
```

We can't assume `\ignorespacesafterend` is defined since it was not there in the earliest releases of LaTeX 2e. And we need to include the `\global` for the same reason.

```
\@ifundefined{ignorespacesafterend}{%
  \def\ignorespacesafterend{\global\@ignoretrue}%
}{}
\newenvironment{subequations}{%
```

Before sending down the 'equation' counter to the subordinate level, add 1 using standard `\refstepcounter`.

```
  \refstepcounter{equation}%
```

Define `\theparentequation` equivalent to current `\theequation`. `\edef` is necessary to expand the current value of the equation counter. This might in rare cases cause something to blow up, in which case the user needs to add `\protect`.

```
  \protected@edef\theparentequation{\theequation}%
  \setcounter{parentequation}{\value{equation}}%
```

And set the equation counter to 0, so that the normal incrementing processes in the various equation environments will produce the desired results.

```
  \setcounter{equation}{0}%
  \def\theequation{\theparentequation\alph{equation}}%
  \ignorespaces
}{%
  \setcounter{equation}{\value{parentequation}}%
  \ignorespacesafterend
}
```

15 Equation numbering

In the multiline equation environments provided here, the task of equation numbering is linked to the task of line breaking in the sense that it is the `\\` command that marks where an equation number for the current line will be processed and added to the page.

Provide a convenient way to specify that equations should be numbered within sections.

```
\newcommand{\numberwithin}[3][\arabic]{%
  \@ifundefined{c@#2}{\@nocounterr{#2}}{%
    \@ifundefined{c@#3}{\@nocnterr{#3}}{%
      \@addtoreset{#2}{#3}%
      \@xp\xdef\csname the#2\endcsname{%
        \@xp\@nx\csname the#3\endcsname .\@nx#1{#2}}}}%
}
```

To make references to equation numbers easier, we provide `\eqref`. We almost don't need `\textup`, except that `\tagform@` doesn't supply the italic correction.

```
\newcommand{\eqref}[1]{\textup{\tagform@{\ref{#1}}}}
```

15.1 Preliminary macros

The following macros implement the LaTeX syntax for the `\\` command, i.e. the possibility to add an asterisk to inhibit a page break, or an optional argument to denote additional vertical space. They are modelled more or less after the corresponding macros for LaTeX's `eqnarray` and `array` environments.

[We can perhaps use the eqnarray mechanism if we change it so that it also uses `\openup`.]

`\dspbrk@lvl` We begin by defining the `\dspbrk@lvl` counter. This counter records the desirability of a break after the current row, as a number between 0 and 4. Its default value is -1 meaning that no explicit `\displaybreak` command was given, and the default `\interdisplaylinepenalty` is to be used.

```
\newcount\dspbrk@lvl
```

```
867 \dspbrk@lvl=-1
```

rdisplaylinepenalty We set the \interdisplaylinepenalty to 10000.

```
868 \interdisplaylinepenalty\@M
```

\allowdisplaybreaks The \allowdisplaybreaks command. Since this is intended for use outside displayed formulas (typically in the preamble), it does not need to use \new@ifnextchar.

```
869 \newcommand{\allowdisplaybreaks}[1][4]{%
870   \interdisplaylinepenalty\getdsp@pen{#1}\relax
871 }
```

\getdsp@pen Modelled after LaTeX's \@getpen. We use higher numbers than would normally be provided by \@lowpenalty, \@medpenalty, and \@highpenalty, since display breaks are almost always less desirable.

```
872 \def\getdsp@pen#1{%
873   \ifcase #1\@M \or 9999 \or 6999 \or 2999 \or \z@\fi
874 }
```

\displaybreak For breaks in a certain row of a alignment.

\dspbrk@

\dspbrk@context

nogood@displaybreak

```
875 \newcommand{\displaybreak}{\new@ifnextchar[\dspbrk@{\dspbrk@[4]}}
876 \chardef\dspbrk@context=\sixt@@n
877 \def\dspbrk@[#1]{%
878   \ifmeasuring@
879   \else
880     \ifcase\dspbrk@context % case 0 --- OK
881       \global\dspbrk@lvl #1\relax
882     \or                    % case 1 --- inside a box
883       \nogood@displaybreak
884     \else                  % other cases --- outside of a display
885       \@amsmath@err{\Invalid@@\displaybreak}\@eha
886     \fi
887   \fi
888 }
```

This is the value of \displaybreak when it occurs inside some structure where it will not work.

```
889 \def\nogood@displaybreak{%
890   \@amsmath@err{\protect
891 \displaybreak\space cannot be applied here}%
892 {One of the enclosing environments creates an
893   unbreakable box\MessageBreak
894 (e.g., split, aligned, gathered, ...).}%
895 }
```

\math@cr The macro \math@cr ends a row inside one of the equation environments, i.e., this is the internal name of the \\ commands in these environments. As usual for this kind of macro inside of alignments we insert a special brace into TeX's input stream. The initial \relax is needed to trigger entry into the u template of the current column if the author ended the current row with an empty column (i.e., the mathcr was immediately preceded by an ampersand).

```
896 \def\math@cr{\relax\iffalse{\fi\ifnum0=`}\fi
```

The first step is now to check whether an asterisk follows. \@eqpen is used to hold the penalty value to be put on the vertical list. Then we call up \math@cr@ which performs the next step. If an asterisk is read page breaking is inhibited.

```
897   \@ifstar{\global\@eqpen\@M\math@cr@}%
```

Otherwise we have to check the \dspbrk@lvl value.

```
          {\global\@eqpen
             \ifnum\dspbrk@lvl <\z@ \interdisplaylinepenalty
              \else -\@getpen\dspbrk@lvl \fi
           \math@cr@}}
```

\math@cr@ The purpose of \math@cr@ is to check whether an optional argument follows. If not it provides \z@ as default value.

```
\def\math@cr@{\new@ifnextchar[\math@cr@@{\math@cr@@[\z@]}}
```

\math@cr@@ \math@cr@@ closes the special brace opened in \math@cr, and calls \math@cr@@@ which is supposed the 'real' row ending command. The meaning of this macro depends on the environment in which it is used.

```
\def\math@cr@@[#1]{\ifnum0=`{\fi \iffalse}\fi\math@cr@@@
```

Finally we put the additional space onto the vertical list.

```
  \noalign{\vskip#1\relax}}
```

\Let@ \Let@ is called by all environments where \\ ends a row of an alignment.

```
\def\Let@{\let\\\math@cr}
```

\restore@math@cr We mentioned already that the exact meaning of \math@cr@@@ depends on the current environment. Since it is often a simple \cr we provide \restore@math@cr to reset it.

```
\def\restore@math@cr{\def\math@cr@@@{\cr}}
```

This is also the default case.

```
\restore@math@cr
```

\intertext \intertext@ The \intertext command is used for inserting text between the rows of an alignment. It might better be done as an environment, but the \begingroup from \begin would cause the \noalign to fail.

```
\newcommand{\intertext}{\@amsmath@err{\Invalid@@\intertext}\@eha}
```

\intertext@ is called by all environments that allow the use of the \intertext command.

```
\def\intertext@{%
  \def\intertext##1{%
```

If current mode is not vmode, the most likely reason is that the writer forgot the \\ that is supposed to precede \intertext. All right, then, let's try adding it our ownself. But, to be slightly careful: \\ does a futurelet, and it's slightly dangerous to allow a letted token to barge around loose in our internal code when it has been let to a conditional token like \fi. So let's interpose something in front of the \fi for the futurelet to take instead. (And careful again: it has to be something evanescent, not (e.g.) \relax which would cause the next halign cell to fire up and keep \noalign from working.)

```
    \ifvmode\else\\\@empty\fi
    \noalign{%
      \penalty\postdisplaypenalty\vskip\belowdisplayskip
      \vbox{\normalbaselines
```

We need to do something extra if the outside environment is a list environment. I don't see offhand an elegant way to test "are we inside any list environment" that is both easy and reliable (for example, checking for zero \@totalleftmargin wouldn't catch the case where \@totalleftmargin is zero but \linewidth is less than \columnwidth), so it seems to me checking \linewidth is the best practical solution.

```
915         \ifdim\linewidth=\columnwidth
916         \else \parshape\@ne \@totalleftmargin \linewidth
917         \fi
918         \noindent##1\par}%
919       \penalty\predisplaypenalty\vskip\abovedisplayskip%
920     }%
921 }}
```

15.2 Implementing tags and labels

In this section we describe some of the macros needed to make the \tag command work in various places. We start by defining a help text to be used when a \tag command is used somewhere it should not appear.

\tag@help This is the default error help text provided when \tag generates an error message. Note that \newhelp generates a control sequence name from the string given as its argument so that a leading backslash is provided automatically.

```
922 \newhelp\tag@help
923   {tag cannot be used at this point.\space
924    If you don't understand why^^Jyou should consult
925    the documentation.^^JBut don't worry: just continue, and I'll
926    forget what happened.}
```

\gobble@tag This macro is to be used when \tag should silently skip its argument. It is made to handle the *-form of \tag as well.

```
927 \def\gobble@tag{\@ifstar\@gobble\@gobble}
```

\invalid@tag \invalid@tag is a macro that should be used whenever \tag appears in an illegal place. It sets up \tag@help (as defined above) as help message, prints its argument as error message, and skips \tag's argument.

```
928 \def\invalid@tag#1{\@amsmath@err{#1}{\the\tag@help}\gobble@tag}
```

\dft@tag \default@tag \dft@tag provides a convenient way to disallow the use of \tag at certain points. One simply has to write

```
\let\tag\dft@tag
```

and the \tag command will produce an error message, with a suitable error help text, and discard its argument.

```
929 \def\dft@tag{\invalid@tag{\string\tag\space not allowed here}}
```

Since this is used several times we provide an abbreviation for it.

```
930 \def\default@tag{\let\tag\dft@tag}
```

Since this is also the default, i.e. the \tag command should not be used except in special places, we issue a \default@tag command.

```
931 \default@tag
```

Now that we have taken care of the case that \tag is not allowed we will provide some macros to process tags appropriately. As the user documentation states, a \tag command (without the asterisk typesets its argument according to the document styles' conventions, whereas a \tag* command typesets its argument exactly as given. We define therefore the following interface:

\maketag@@ \maketag@@@ \tagform@ \tag is supposed to call \maketag@@ which checks whether an asterisk follows. If this is the case it calls up \maketag@@@ which sets its argument 'as is'. Otherwise \tagform@ is called to do the job. (This macro is to be defined appropriately by the document style.)

```
932 \def\maketag@@{\@ifstar\maketag@@@\tagform@}
```

We define \maketag@@@ to use the normal font of the document text (since this is the usual practice for numbering of document elements) and to put a box around the tag. Furthermore we use \m@th for exceptional cases where the tag involves a superscript or some such math. (Probably from an explicit use of \tag* rather than from the automatic numbering.)

```
\def\maketag@@@#1{\hbox{\m@th\normalfont#1}}
```

We use the following default definition for \tagform@ that puts only parentheses around the tag.

```
\def\tagform@#1{\maketag@@@{(\ignorespaces#1\unskip\@@italiccorr)}}
```

We need to insinuate \tagform@ into \@eqnnum in case eqnarray is used (probably in a document that was originally written without use of the amsmath package).

```
\iftagsleft@
  \def\@eqnnum{\hbox to1sp{}\rlap{\normalfont\normalcolor
    \hskip -\displaywidth\tagform@\theequation}}
\else
  \def\@eqnnum{{\normalfont\normalcolor \tagform@\theequation}}
\fi
```

\thetag Sometimes one needs to set a literal tag according to the rules of the document style. To achieve this we provide the \thetag command. It typesets its argument by calling \tagform@ on it.

```
\newcommand{\thetag}{\leavevmode\tagform@}
```

\df@tag \make@df@tag \make@df@tag@@ \make@df@tag@@@ Sometimes it is necessary for a \tag command to store a tag in a safe place and to process it later, e.g., for a tag in a row of an alignment where the tag can only be typeset when the \\ at the end of the row was seen. Such a tag is stored in the macro \df@tag (for 'deferred tag'). For this purpose we provide the \make@df@tag macro. It is built very similar to the \maketag@@ macro above.

```
\let\df@tag\@empty
\def\make@df@tag{\@ifstar\make@df@tag@@\make@df@tag@@@}
```

\make@df@tag sets \@currentlabel and defines \df@tag appropriately.

To simplify the task of tracking \tag and \label commands inside math display environments, we defer \label commands until the tag is typeset, similar to the way that \tags themselves are deferred. This allows arbitrary placement of \label and \tag commands and also means we only increment the \equation counter when we really need to, thus avoiding the \setb@ck nonsense that used to be required.

```
\def\make@df@tag@@#1{%
  \gdef\df@tag{\maketag@@@{#1}\def\@currentlabel{#1}}}
```

Autogenerated number:

```
\def\make@df@tag@@@#1{\gdef\df@tag{\tagform@{#1}%
  \toks@\@xp{\p@equation{#1}}\edef\@currentlabel{\the\toks@}}}
```

\ltx@label \label@in@display \df@label Next, we store the default definition of \label in \ltx@label and then define a new version of \label for use in math display environments. \label@in@display merely issues a warning message if there is already a pending label (which will be discarded) and then stores the label in \df@label.

```
\let\ltx@label\label
%
\def\label@in@display{%
    \ifx\df@label\@empty\else
```

```
        \@amsmath@err{Multiple \string\label's:
            label '\df@label' will be lost}\@eha
    \fi
    \gdef\df@label
}
```

In case there is an enumerate inside a minipage inside an equation, we need to reset `\label` to its normal value:

```
\toks@\@xp{\@arrayparboxrestore \let\label\ltx@label}%
\edef\@arrayboxrestore{\the\toks@}
\let\df@label\@empty
```

`\make@display@tag` Now we define a macro to process `\tag` and `\label` commands in various display environments. If the `@eqnsw` switch is set, then we should supply an equation number; otherwise, if the `@tag` switch is set, we should use the tag stored in `\df@tag`. Finally, we process any pending `\label`s.

TODO: Arguably, `\make@display@tag` should issue a warning message if there is a `\label` but neither a tag nor an equation number. Also, it would probably be worthwhile to explore whether `\iftag@` could be done away with and replaced by checks to see if `\df@tag` is empty or not.

```
\def\make@display@tag{%
  \if@eqnsw \incr@eqnum \print@eqnum
  \else \iftag@ \df@tag \global\let\df@tag\@empty \fi
  \fi
```

Need to check the `\ifmeasuring@` flag otherwise the `\write` node from `\label` might be discarded in a temp box and clearing `\df@label` will keep it from being reiterated on the real typesetting pass.

```
  \ifmeasuring@
  \else
    \ifx\df@label\@empty
    \else
      \@xp\ltx@label\@xp{\df@label}%
      \global\let\df@label\@empty
    \fi
  \fi
}
```

Now we define the special versions of `\tag` used within the `align` environments.

`\tag@in@align` The `\tag` command may only appear once in a row of an alignment. Therefore we first check the switch `tag@` that is set to false at the begin of every row. If this switch is true a `\tag` was already given in this row and we define `\next@` to expand to a call to `\invalid@tag`.

```
\def\tag@in@align{%
    \relax
    \iftag@
        \DN@{\invalid@tag{Multiple \string\tag}}%
    \else
```

Otherwise we set the `tag@` switch. But there is more to be done: we must also prevent the automatic generation of a tag. Therefore we also reset the `@eqnsw`.

```
    \global\tag@true
```

Changed to `\nonumber`, since that seems to be all that's required.—dmj, 1994/12/21

```
    \nonumber
```

Within a row of an `align` environment the `\tag` command must not typeset the tag immediately since its position can be determined only later. Therefore we use the `\make@df@tag` macro defined earlier. Finally we call `\next@` to process the argument that follows.

```
        \let\next@\make@df@tag
    \fi
    \next@
}
```

`\raisetag` Usage: `\raisetag` ⟨*dimen*⟩

This will modify the vertical placement of the tag of the current equation by ⟨*dimen*⟩. Note that according to the current uses of `\raise@tag` in e.g., `\place@tag@gather`, no adjustment occurs if the tag falls in its normal position; i.e., `\raisetag` has no effect unless the tag has already been shifted off-line.

```
\newcommand{\raisetag}[1]{\skip@#1\relax
  \xdef\raise@tag{\vskip\iftagsleft@\else-\fi\the\skip@\relax}%
}
```

`\raise@tag` will be reemptied at the beginning of each equation, which might occur at a `\begin{xxx}` or `\\`.

```
\let\raise@tag\@empty
```

`\notag` For consistency we provide `\notag`, equivalent to `\nonumber`. The alternative would have been to rename `\tag` as `\number` to go along with `\nonumber`, but of course `\number` is a TEX primitive that should not be redefined.

```
\newcommand{\notag}{\nonumber}
```

`\nonumber` Need to add some additional code to `\nonumber` to deal with some complications related to nested environments.

```
\renewcommand{\nonumber}{%
  \if@eqnsw
    \ifx\incr@eqnum\@empty \addtocounter{equation}\m@ne \fi
  \fi
  \let\print@eqnum\@empty \let\incr@eqnum\@empty
  \global\@eqnswfalse
}
```

```
\def\print@eqnum{\tagform@\theequation}
\def\incr@eqnum{\refstepcounter{equation}\let\incr@eqnum\@empty}
```

16 Multiline equation environments

16.1 Remarks

In late 1994 David M. Jones did a thorough overhaul of these environments so that the number placement and a few other aspects are substantially improved over the original versions that were ported essentially unchanged from `amstex.tex` in 1989. Most of the commentary in this section is DMJ's, and comments of any significance that I added are marked by my initials and date [mjd, 1995/01/11].

16.2 Preliminaries

`\ifinalign@` `\ifingather@` We define two switches that are set to true in certain alignments: `inalign@` and `ingather@` inside of the `align` and `gather` environments. These switches are needed to control certain actions that depend on the surrounding conditions, more specifically: on the setting already done by the surrounding environments.

```
\newif\ifinalign@
\newif\ifingather@
```

Historical Note: Removed the `\ifinany@` test [mjd,1999/06/28] since it was mainly used for the purpose now handled by `\spread@equation`.

`\@arrayparboxrestore` Here we must reset a few additional parameters.

```
\@xp\def\@xp\@arrayparboxrestore\@xp{\@arrayparboxrestore
  \ingather@false\inalign@false \default@tag
  \let\spread@equation\@spread@equation
  \let\reset@equation\@empty
  \def\print@eqnum{\tagform@\theequation}%
  \def\incr@eqnum{\refstepcounter{equation}\let\incr@eqnum\@empty}%
}
```

`\iftag@` The switch `tag@` is set to false at the beginning of every row and set to true by a `\tag` command. This allows us to check whether there is more than one tag on a row.

```
\newif\iftag@
```

`\ifst@rred` The switch `st@rred` is set to true by all starred environments and set to false by the unstarred versions.

```
\newif\ifst@rred
```

`\ifmeasuring@` All display environments get typeset twice—once during a "measuring" phase and then again during a "production" phase; `\ifmeasuring@` will be used to determine which case we're in, so we can take appropriate action.

```
\newif\ifmeasuring@
```

`\ifshifttag@` `\ifshifttag@` is used by `gather` to communicate between `\calc@shift@gather` and `\place@tag@gather` whether an equation tag should be shifted to a separate line. It's also used by `multline`.

```
\newif\ifshifttag@
```

`\row@`

```
\newcount\row@
```

`\column@` The counter `\column@` is used by the alignment macros to keep track of the current column.

```
\newcount\column@
```

`\column@plus` `\column@plus` is a useful abbreviation.

```
\def\column@plus{%
    \global\advance\column@\@ne
}
```

`\maxfields@`

```
\newcount\maxfields@
```

`\add@amp`
`\add@amps`

```
\def\add@amp#1{\if m#1&\@xp\add@amp\fi}
\def\add@amps#1{%
    \begingroup
    \count@#1\advance\count@-\column@
    \edef\@tempa{\endgroup
      \@xp\add@amp\romannumeral\number\count@ 000q}%
    \@tempa
}
```

\andhelp@ The help text stored in \andhelp@ is used for errors generated by too many & characters in a row.

```
\newhelp\andhelp@
{An extra & here is so disastrous that you should probably exit^^J
and fix things up.}
```

\eqnshift@ \eqnshift@ is used by align and gather as the indentation of the lines of the environment from the left margin.

```
\newdimen\eqnshift@
```

\alignsep@

```
\newdimen\alignsep@
```

\tagshift@

```
\newdimen\tagshift@
```

\mintagsep \mintagsep is the minimum allowable separation between an equation and its tag. We set it to half a quad in \textfont2, which is TEX's built-in value.

```
\newcommand{\mintagsep}{.5\fontdimen6\textfont\tw@}
```

\minalignsep This should probably be a skip register [mjd,1999/06/18]

```
\newcommand{\minalignsep}{10pt}
```

\tagwidth@

```
\newdimen\tagwidth@
```

\totwidth@

```
\newdimen\totwidth@
```

\lineht@ The dimen register \lineht@ is used to keep track of the height (or depth, if tags are on the right) of a row in an alignment.

```
\newdimen\lineht@
```

\tag@width
\savetaglength@
\shift@tag
\tag@shifts

```
\def\tag@width#1{%
    \ifcase\@xp#1\tag@lengths\fi
}

\def\savetaglength@{%
    \begingroup
        \let\or\relax
        \xdef\tag@lengths{\tag@lengths\or \the\wdz@}%
    \endgroup
}

\def\shift@tag#1{%
    \ifcase\@xp#1\tag@shifts\fi\relax
}

\let\tag@shifts\@empty
```

\saveshift@

```
\def\saveshift@#1{%
    \begingroup
        \let\or\relax
        \xdef\tag@shifts{\or#1\tag@shifts}%
    \endgroup
}
```

\spread@equation This does the line-spacing adjustment that is normally wanted for displayed equations. We also call \reset@strutbox@ here because otherwise a preceding font size change might leave \strutbox@ with wrong contents. This is a less-than-ideal solution but probably good enough for now, until the situation can be overhauled.

```
\def\spread@equation{\reset@strutbox@
  \openup\jot \let\spread@equation\@empty}
\let\@spread@equation\spread@equation
```

\displ@y \displ@y@ \@display@init \displ@y is from plain.tex, with \interdisplaylinepenalty changed to \@eqpen. Also we transplanted most of its internal organs to \@display@init to support \displ@y@ and other possibilities. Don't try to make sense of these naming conventions! They are a narrowly calculated mishmash of Knuth/Spivak/Lamport/Mittelbach precedents. The reason for not cleaning them up and forcing all names to a consistent scheme is that then in principle we'd have to do it everywhere else too. And we programmers are paranoid about the side effects of name changes.

```
\def\displ@y{\@display@init{}}
\def\@display@init#1{%
    \global\dt@ptrue \spread@equation
    \everycr{%
        \noalign{%
            #1%
            \ifdt@p
                \global\dt@pfalse
                \vskip-\lineskiplimit
                \vskip\normallineskiplimit
            \else
                \penalty\@eqpen \global\dspbrk@lvl\m@ne
            \fi
        }%
    }%
}
```

\displ@y@ is nearly the same; it additionally sets the tag@ switch and the \column@ and \dspbrk@lvl counters to their default values. The argument is normally a bit of code to empty out \raise@tag, but in multline we don't want that to happen in \everycr.

```
\def\displ@y@{\@display@init{%
  \global\column@\z@ \global\dspbrk@lvl\m@ne
  \global\tag@false \global\let\raise@tag\@empty
}}
```

\black@ This macro is made to produce an overfull box message and possibly (depending on the value of \overfullrule) a rule in the margin if the total width of an alignment is larger than the value of \displaywidth.

```
\def\black@#1{%
    \noalign{%
        \ifdim#1>\displaywidth
            \dimen@\prevdepth
            \nointerlineskip
            \vskip-\ht\strutbox@
            \vskip-\dp\strutbox@
            \vbox{\noindent\hbox to#1{\strut@\hfill}}%
            \prevdepth\dimen@
        \fi
    }%
}
```

\savecounters@ \restorecounters@

These are used during the measuring phase of the various display math environments to save and restore the values of all LaTeX counters. We make these local to a group, so nested environments works.

Changed \stepcounter to \csname c@...\endcsname to avoid overhead of ifundefined test [mjd, 1995/01/20].

```
\def\savecounters@{%
    \begingroup
        \def\@elt##1{%
          \global\csname c@##1\endcsname\the\csname c@##1\endcsname}%
        \xdef\@gtempa{%
            \cl@@ckpt
            \let\@nx\restorecounters@\@nx\@empty
        }%
    \endgroup
    \let\restorecounters@\@gtempa
}
%
\let\restorecounters@\@empty
```

\savealignstate@ \restorealignstate@

These are used to save the values of various parameters that are shared by align and gather when the former is used inside the latter.

```
\def\savealignstate@{%
    \begingroup
        \let\or\relax
        \xdef\@gtempa{%
            \global\totwidth@\the\totwidth@
            \global\row@\the\row@
            \gdef\@nx\tag@lengths{\tag@lengths}%
            \let\@nx\restorealignstate@\@nx\@empty
        }%
    \endgroup
    \let\restorealignstate@\@gtempa
}

\let\restorealignstate@\@empty
```

\savecolumn@ \restorecolumn@

```
\def\savecolumn@{%
  \edef\restorecolumn@{%
    \global\column@\number\column@
    \let\@nx\restorecolumn@\@nx\@empty
  }%
}
\let\restorecolumn@\@empty
```

16.3 Scanning the environment's body

Several of the math alignment macros must scan their body twice: once to determine how wide the columns are and then to actually typeset them. This means that we must collect all text in this body before calling the environment macros.

\@envbody

We start by defining a token register to contain the body.

```
\newtoks\@envbody
```

\addto@envbody

Then we define a macro to add something (i.e. its argument) to the token register \@envbody.

```
\def\addto@envbody#1{\global\@envbody\@xp{\the\@envbody#1}}
```

\collect@body The macro \collect@body starts the scan for the \end{...} command of the current environment. It takes a macro name as argument. This macro is supposed to take the whole body of the environment as its argument. For example, \begin{align} would call \collect@body\@align if @align#1{...} is the macro that sets the alignment with body #1.

```
\def\collect@body#1{%
  \@envbody{\@xp#1\@xp{\the\@envbody}}%
  \edef\process@envbody{\the\@envbody\@nx\end{\@currenvir}}%
  \@envbody\@emptytoks \def\begin@stack{b}%
```

If we simply called \collect@@body directly, the error message for a \par token (usually from a blank line) would be

```
! Paragraph ended before \collect@@body was complete.
```

But we use a little finesse to get a more intelligible error message:

```
! Paragraph ended before \multline* was complete.
```

In order to avoid using up csnames unnecessarily we use the actual environment name as the name of the temporary function that is \let to \collect@@body; but then in order to preserve the theoretical possibility of nesting for environments that use \collect@body (not currently required by any amsmath environment [mjd,1999/06/23]), we do the \let inside a group.

```
  \begingroup
  \@xp\let\csname\@currenvir\endcsname\collect@@body
```

This small twist eliminates the need for \expandafter's in \collect@@body.

```
  \edef\process@envbody{\@xp\@nx\csname\@currenvir\endcsname}%
  \process@envbody
}
```

\push@begins When adding a piece of the current environment's contents to \@envbody, we scan it to check for additional \begin tokens, and add a 'b' to the stack for any that we find.

```
\def\push@begins#1\begin#2{%
  \ifx\end#2\else b\@xp\push@begins\fi
}
```

\collect@@body \collect@@body takes two arguments: the first will consist of all text up to the next \end command, the second will be the \end command's argument. If there are any extra \begin commands in the body text, a marker is pushed onto a stack by the \push@begins function. Empty state for this stack means that we have reached the \end that matches our original \begin. Otherwise we need to include the \end and its argument in the material that we are adding to our environment body accumulator.

Historical Note: In a former implementation, the error messages resulting from a typo in the environment name were unsatisfactory, because it was matching of the environment name that was used to determine the end of our environment body, instead of counting begin-end pairs. Thanks to Lars Hellström for a suggestion that led to this improvement. [mjd,1999/06/23]

```
\def\collect@@body#1\end#2{%
  \edef\begin@stack{\push@begins#1\begin\end \@xp\@gobble\begin@stack}%
  \ifx\@empty\begin@stack
    \endgroup
```

```
    \@checkend{#2}%
    \addto@envbody{#1}%
  \else
    \addto@envbody{#1\end{#2}}%
  \fi
  \process@envbody % A little tricky! Note the grouping
}
```

16.4 Simple aligning environments

\math@cr@@@aligned From tabskip we get an extra space of minalignsep after every second column; but when this falls at the right edge of the whole aligned structure, we need to cancel that space.

```
\def\math@cr@@@aligned{%
  \ifodd\column@ \let\next@\@empty
  \else \def\next@{&\kern-\alignsep@}%
  \fi
  \next@ \cr
}
```

\start@aligned The `aligned` and `alignedat` environments are identical except that the latter takes a mandatory argument to specify the number of align structures, while the former allows any number of align structures automatically (the use of `alignedat` is deprecated). So, they will be defined in terms of `\start@aligned`, which will take two arguments. The first argument specifies the placement of the environments; it is either `c`, `t`, or `b`. The second is the number of align structures; a value of -1 means that an arbitrary number are allowed.

```
\newcommand{\start@aligned}[2]{%
    \RIfM@\else
        \nonmatherr@{\begin{\@currenvir}}%
    \fi
    \savecolumn@ % Assumption: called inside a group
```

The `\null` here is to keep the `\,` glue from causing the invocation of the clause in TEX's built-in tag placement algorithm that can cause an equation to be shifted all the way over to the margin.

```
    \null\,%
    \if #1t\vtop \else \if#1b \vbox \else \vcenter \fi \fi \bgroup
        \maxfields@#2\relax
        \ifnum\maxfields@>\m@ne
            \multiply\maxfields@\tw@
```

Introduced new `\math@cr@@@` so we can provide standard error message for too many `&`'s in `alignedat`.

```
            \let\math@cr@@@\math@cr@@@alignedat
            \alignsep@\z@skip
        \else
            \let\math@cr@@@\math@cr@@@aligned
            \alignsep@\minalignsep
        \fi
```

Reset the meaning of `\\`.

```
        \Let@ \chardef\dspbrk@context\@ne
```

Restore the default definition of `\tag` (error message), in case `aligned` is used inside, e.g., a `gather` environment that accepts `\tag`.

```
        \default@tag
        \spread@equation % no-op if already called
```

Finally we start the alignment itself. For `aligned` we add `\minalignsep` after every second column to mimic the behavior of `align`. For `alignedat` the user has to specify interalign space explicitly.

```
        \global\column@\z@
        \ialign\bgroup
           &\column@plus
            \hfil
            \strut@
            $\m@th\displaystyle{##}$%
            \tabskip\z@skip
           &\column@plus
            $\m@th\displaystyle{{}##}$%
            \hfil
            \tabskip\alignsep@
            \crcr
}
```

`\math@cr@@@aligned` `\math@cr@@@aligned` checks to make sure the user hasn't put in too many `&`s in `alignedat`. Since `alignedat` doesn't use `\displ@y@`, we also reset `\column@` here. Note than in `aligned`, `\column@` will increase without bound, since it never gets reset, but this is harmless.

```
\def\math@cr@@@alignedat{%
    \ifnum\column@>\maxfields@
        \begingroup
          \measuring@false
          \@amsmath@err{Extra & on this line}%
            {\the\andhelp@}% "An extra & here is disastrous"
        \endgroup
    \fi
    \global\column@\z@
    \cr
}
```

`\alignsafe@testopt` Testing for an optional argument can be really, really tricky in certain complicated contexts. This we discovered by getting some bug reports for uses of `aligned`. So here is a safer form of LaTeX's `\@testopt` function.

```
\def\alignsafe@testopt#1#2{%
  \relax\iffalse{\fi\ifnum`}=0\fi
  \@ifnextchar[%
    {\let\@let@token\relax \ifnum`{=\z@\fi\iffalse}\fi#1}%
    {\let\@let@token\relax \ifnum`{=\z@\fi\iffalse}\fi#1[#2]}%
}
```

`aligned` The `aligned` environment takes an optional argument that indicates its vertical position in relation to surrounding material: `t`, `c`, or `b` for top, center, or bottom.

```
\newenvironment{aligned}{%
  \let\@testopt\alignsafe@testopt
  \aligned@a
}{%
  \crcr\egroup
  \restorecolumn@
  \egroup
}
\newcommand{\aligned@a}[1][c]{\start@aligned{#1}\m@ne}
```

`alignedat` To get a top or bottom positioned `alignedat` structure, you would write something like

```
\begin{alignedat}[t]{3}
```

```
\newenvironment{alignedat}{%
  \let\@testopt\alignsafe@testopt
  \alignedat@a
}{%
  \endaligned
}
\newcommand{\alignedat@a}[1][c]{\start@aligned{#1}}
```

gathered The gathered environment is for several lines that are centered independently.

```
\newenvironment{gathered}[1][c]{%
    \RIfM@\else
        \nonmatherr@{\begin{gathered}}%
    \fi
    \null\,%
    \if #1t\vtop \else \if#1b\vbox \else \vcenter \fi\fi \bgroup
        \Let@ \chardef\dspbrk@context\@ne \restore@math@cr
        \spread@equation
        \ialign\bgroup
            \hfil\strut@$\m@th\displaystyle##$\hfil
            \crcr
}{%
  \endaligned
}
```

16.5 The gather environment

\start@gather

```
\def\start@gather#1{%
    \RIfM@
        \nomath@env
        \DN@{\@namedef{end\@currenvir}{}\@gobble}%
    \else
        $$%
        #1%
        \ifst@rred \else \global\@eqnswtrue \fi
        \let\next@\gather@
    \fi
    \collect@body\next@
}
```

gather
gather*

```
\newenvironment{gather}{%
  \start@gather\st@rredfalse
}{%
  \math@cr \black@\totwidth@ \egroup
  $$\ignorespacesafterend
}

\newenvironment{gather*}{%
  \start@gather\st@rredtrue
}{%
  \endgather
}
```

\gather@

```
\def\gather@#1{%
    \ingather@true \let\split\insplit@
```

```
    \let\tag\tag@in@align \let\label\label@in@display
    \chardef\dspbrk@context\z@
    \intertext@ \displ@y@ \Let@
    \let\math@cr@@@\math@cr@@@gather
    \gmeasure@{#1}%
    \global\shifttag@false
    \tabskip\z@skip
    \global\row@\@ne
    \halign to\displaywidth\bgroup
        \strut@
        \setboxz@h{$\m@th\displaystyle{##}$}%
        \calc@shift@gather
        \set@gather@field
        \tabskip\@centering
       &\setboxz@h{\strut@{##}}%
        \place@tag@gather
        \tabskip \iftagsleft@ \gdisplaywidth@ \else \z@skip \span\fi
        \crcr
        #1%
}
```

`\gmeasure@`

```
\def\gmeasure@#1{%
    \begingroup
        \measuring@true
        \totwidth@\z@
        \global\let\tag@lengths\@empty
        \savecounters@
        \setbox\@ne\vbox{%
            \everycr{\noalign{\global\tag@false
              \global\let\raise@tag\@empty \global\column@\z@}}%
            \let\label\@gobble
            \halign{%
                \setboxz@h{$\m@th\displaystyle{##}$}%
                \ifdim\wdz@>\totwidth@
                    \global\totwidth@\wdz@
                \fi
               &\setboxz@h{\strut@{##}}%
                \savetaglength@
                \crcr
                #1%
                \math@cr@@@
            }%
        }%
        \restorecounters@
        \if@fleqn
            \global\advance\totwidth@\@mathmargin
        \fi
        \iftagsleft@
            \ifdim\totwidth@>\displaywidth
                \global\let\gdisplaywidth@\totwidth@
            \else
                \global\let\gdisplaywidth@\displaywidth
            \fi
        \fi
    \endgroup
}
```

`\math@cr@@@gather` Modified `\math@cr@@@gather` so that it always puts in the final field, which

needs to be done under the new method for determining tag placement. This is probably more efficient anyway.

```
\def\math@cr@@@gather{%
    \ifst@rred\nonumber\fi
   &\relax
    \make@display@tag
    \ifst@rred\else\global\@eqnswtrue\fi
```

We advance `\row@` here, rather than at the beginning of the preamble, because otherwise the `split` environment will cause `\row@` to be advanced twice instead of once.

```
    \global\advance\row@\@ne
    \cr
}
```

`\calc@shift@gather` `\calc@shift@gather` has must make two decisions: (1) whether the equation tag for the current line should be put on a separate line and (2) what the distance between the equation and the equation tag should be. We implement TEX's built-in tag-placement as well as possible, with one improvement: the minimum separation between tag and equation is now a user-settable parameter.

[1995/01/17] Added a check to make sure that the width of the tag on the current line is $>$ 0 before testing to see if tagwidth + linewidth + mintagsep $>$ displaywidth. Since an imbedded align shows up as line with width `\displaywidth`, and even lines without a tag get processed as if an empty tag were present, the result was that the empty tag assigned to the line containing the align was being shifted downwards, creating extra space after the align.

```
\def\calc@shift@gather{%
    \dimen@\mintagsep\relax
    \tagwidth@\tag@width\row@\relax
```

If we're in `fleqn` mode, there is no flexibility about placement of the equation, so all we can do is see if there's room for the tag in the given margin.

```
    \if@fleqn
        \global\eqnshift@\@mathmargin
        \ifdim\tagwidth@>\z@
            \advance\dimen@\tagwidth@
            \iftagsleft@
                \ifdim\dimen@>\@mathmargin
                    \global\shifttag@true
                \fi
            \else
                \advance\dimen@\@mathmargin
                \advance\dimen@\wdz@
                \ifdim\dimen@>\displaywidth
                   \global\shifttag@true
                \fi
            \fi
        \fi
    \else
        \global\eqnshift@\displaywidth
        \global\advance\eqnshift@-\wdz@
        \ifdim\tagwidth@>\z@
            \multiply\dimen@\tw@
            \advance\dimen@\wdz@
            \advance\dimen@\tagwidth@
            \ifdim\dimen@>\displaywidth
                \global\shifttag@true
            \else
```

```
                \ifdim\eqnshift@<4\tagwidth@
                    \global\advance\eqnshift@-\tagwidth@
                \fi
            \fi
        \fi
        \global\divide\eqnshift@\tw@
        \iftagsleft@
            \global\eqnshift@-\eqnshift@
            \global\advance\eqnshift@\displaywidth
            \global\advance\eqnshift@-\wdz@
        \fi
        \ifdim\eqnshift@<\z@
            \global\eqnshift@\z@
        \fi
    \fi
}
```

\place@tag@gather
\set@gather@field

```
\def\place@tag@gather{%
    \iftagsleft@
        \kern-\gdisplaywidth@
        \ifshifttag@
            \rlap{\vbox{%
                \normalbaselines
                \boxz@
                \vbox to\lineht@{}%
                \raise@tag
            }}%
            \global\shifttag@false
        \else
            \rlap{\boxz@}%
        \fi
    \else
        \ifdim\totwidth@>\displaywidth
            \dimen@\totwidth@
            \advance\dimen@-\displaywidth
            \kern-\dimen@
        \fi
        \ifshifttag@
            \llap{\vtop{%
                \raise@tag
                \normalbaselines
                \setbox\@ne\null
                \dp\@ne\lineht@
                \box\@ne
                \boxz@
            }}%
            \global\shifttag@false
        \else
            \llap{\boxz@}%
        \fi
    \fi
}
%
\def\set@gather@field{%
    \iftagsleft@
        \global\lineht@\ht\z@
    \else
        \global\lineht@\dp\z@
```

```
    \fi
    \kern\eqnshift@
    \boxz@
    \hfil
}
```

16.6 The `align` family of environments

The `align`, `flalign`, `alignat`, `xalignat`, and `xxalignat` environments are virtually identical, and thus will share much code. We'll refer to the environments generically as "`align`" and will distinguish between them explicitly only when necessary.

`\ifxxat@` `\ifcheckat@` `\xatlevel@`

The `\xatlevel@` macro will be used, informally speaking, to distinguish between the `alignat` and `xalignat`, and `xxalignat` environments.

```
\newif\ifxxat@

\newif\ifcheckat@

\let\xatlevel@\@empty
```

`\start@align`

`\start@align` will be called by all of the `align`-like environments. The first argument will be the `\xatlevel@`, i.e., 0, 1, or 2; the second argument will be either `\st@rredtrue` or `\st@rredfalse`. The third argument will be the number of aligned structures in the environment (either as supplied by the user, or -1 to indicate that checking shouldn't be done). After performing the appropriate error detection and initialization, `\start@align` calls `\align@`.

Note that the `\equation` counter is no longer stepped at the beginning of these environments.

TODO: Implement `\shoveleft` and `\shoveright` for `align`.

```
\def\start@align#1#2#3{%
    \let\xatlevel@#1% always \z@, \@ne, or \tw@
    \maxfields@#3\relax
    \ifnum\maxfields@>\m@ne
        \checkat@true
        \ifnum\xatlevel@=\tw@
            \xxat@true
        \fi
        \multiply\maxfields@\tw@
    \else
        \checkat@false
    \fi
    \ifingather@
        \iffalse{\fi\ifnum0=`}\fi
        \DN@{\vcenter\bgroup\savealignstate@\align@#2}%
    \else
        \ifmmode
          \if@display
             \DN@{\align@recover}%
          \else
            \nomath@env
            \DN@{\@namedef{end\@currenvir}{}\@gobble}%
          \fi
        \else
            $$%
            \let\split\insplit@
            \DN@{\align@#2}%
        \fi
```

```
    \fi
    \collect@body\next@
}
```

With version 1.2 of `amsmath`, it was possible to use `align*` and relatives in certain wrong contexts without getting an error, e.g.

```
\begin{equation*}
\begin{align*}
...
\end{align*}
\end{equation*}
```

For backward compatibility we therefore give only a warning for this condition instead of a full error, and try to recover using the `aligned` environment. The alignment of the material may be adversely affected but it will at least remain readable.

```
\def\align@recover#1#2#3{%
  \endgroup
  \@amsmath@err{%
Erroneous nesting of equation structures;\MessageBreak
trying to recover with `aligned'%
  }\@ehc
  \begin{aligned}\relax#1\end{aligned}%
}
```

align
align*
flalign
flalign*
alignat
alignat*
xalignat
xalignat*
xxalignat

The definitions of the various `align` environments are quite straight-forward.

```
\newenvironment{alignat}{%
  \start@align\z@\st@rredfalse
}{%
  \endalign
}
\newenvironment{alignat*}{%
  \start@align\z@\st@rredtrue
}{%
  \endalign
}
\newenvironment{xalignat}{%
  \start@align\@ne\st@rredfalse
}{%
  \endalign
}
\newenvironment{xalignat*}{%
  \start@align\@ne\st@rredtrue
}{%
  \endalign
}
\newenvironment{xxalignat}{%
  \start@align\tw@\st@rredtrue
}{%
  \endalign
}
\newenvironment{align}{%
  \start@align\@ne\st@rredfalse\m@ne
}{%
  \math@cr \black@\totwidth@
  \egroup
  \ifingather@
    \restorealignstate@
```

```
    \egroup
    \nonumber
    \ifnum0=`{\fi\iffalse}\fi
  \else
    $$%
  \fi
  \ignorespacesafterend
}
\newenvironment{align*}{%
  \start@align\@ne\st@rredtrue\m@ne
}{%
  \endalign
}
\newenvironment{flalign}{%
  \start@align\tw@\st@rredfalse\m@ne
}{%
  \endalign
}
\newenvironment{flalign*}{%
  \start@align\tw@\st@rredtrue\m@ne
}{%
  \endalign
}
```

`\align@` TODO: Some of these sets of initializations show up in multiple places. It might be worth making an abbreviation for them.

```
\def\align@#1#2{%
    \inalign@true \intertext@ \Let@ \chardef\dspbrk@context\z@
    \ifingather@\else\displ@y@\fi
    \let\math@cr@@@\math@cr@@@align
    \ifxxat@\else \let\tag\tag@in@align \fi
    \let\label\label@in@display
    #1% set st@r
    \ifst@rred\else \global\@eqnswtrue \fi
    \measure@{#2}%
    \global\row@\z@
    \tabskip\eqnshift@
    \halign\bgroup
        \span\align@preamble\crcr
        #2%
}
```

`\math@cr@@@align`

```
\def\math@cr@@@align{%
  \ifst@rred\nonumber\fi
  \if@eqnsw \global\tag@true \fi
  \global\advance\row@\@ne
  \add@amps\maxfields@
  \omit
  \kern-\alignsep@
  \iftag@
    \setboxz@h{\@lign\strut@{\make@display@tag}}%
    \place@tag
  \fi
  \ifst@rred\else\global\@eqnswtrue\fi
  \global\lineht@\z@
  \cr
}
```

\@cr@@@align@measure

```
\def\math@cr@@@align@measure{%
   &\omit
    \global\advance\row@\@ne
    \ifst@rred\nonumber\fi
    \if@eqnsw \global\tag@true \fi
    \ifnum\column@>\maxfields@
        \ifcheckat@
            \begingroup
              \measuring@false
              \@amsmath@err{Extra & on this line}%
                {\the\andhelp@}% "An extra & here is disastrous"
            \endgroup
        \else
            \global\maxfields@\column@
        \fi
    \fi
    \setboxz@h{\@lign\strut@{%
        \if@eqnsw
            \stepcounter{equation}%
            \tagform@\theequation
        \else
            \iftag@\df@tag\fi
        \fi
    }}%
    \savetaglength@
    \ifst@rred\else\global\@eqnswtrue\fi
    \cr
}
```

\field@lengths
\savefieldlength@
\fieldlengths@

```
\let\field@lengths\@empty

\def\savefieldlength@{%
    \begingroup
        \let\or\relax
        \xdef\field@lengths{%
            \field@lengths
            \ifnum\column@=0
                \or
            \else
                ,%
            \fi
            \the\wdz@
        }%
    \endgroup
}

\def\fieldlengths@#1{%
    \ifcase\@xp#1\field@lengths\fi
}
```

\maxcolumn@widths

\maxcolumn@widths will be used to hold the widths of the fields of the alignat environment. The widths will be separated by the token \or, making it easy to extract a given width using \ifcase.

```
\let\maxcolumn@widths\@empty
```

\maxcol@width

\maxcol@width n = maximum width of nth column of the current alignat (i.e., the nth field of \maxcolumn@widths.) It expands to a ⟨*dimen*⟩, so it can be used

as the right-hand side of a ⟨*variable assignment*⟩ or ⟨*arithmetic*⟩ statement. It's argument can be any ⟨*number*⟩, ⟨*integer variable*⟩ or macro that expands to one of these. [Check to make sure this is true.]

This is subtler than it looks.

```
\def\maxcol@width#1{%
    \ifcase\@xp#1\maxcolumn@widths\fi\relax
}
```

Now comes the real fun. A typical `align` environments looks something like this, where the vertical bars mark the edges of the fields of the underlying `\halign`:

$$\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ \left|\begin{aligned} V_i + q_i v_j &= v_i, \\ V_j &= v_j, \end{aligned}\right. & \left|\begin{aligned} X_i &= x_i - q_i x_j, \\ X_j &= x_j, \end{aligned}\right. & \left|\begin{aligned} U_i &= u_i, \qquad \text{for } i \neq j; \\ U_j &= u_j + \sum_{i \neq j} q_i u_i. \end{aligned}\right. \end{array} \tag{3, 4}$$

Note that each align structure consists of two fields, with no space between them (a small space has been added here to highlight the boundaries). Furthermore, the text inside the odd-numbered fields is flushright, while the text inside the even-numbered fields is flushleft. The equation tags (shown on the right here) can be on either the right or the left. If there is not room (in a sense to be defined shortly) for the tag on the same line as the equation, the tag will be shifted to a separate line.

Each environment also has a certain number of "flexible spaces," meaning spaces whose width we are allowed to adjust to take up the amount of "free space" in the line, meaning the space not taken up by the equation tag and the fields of the underlying `\halign`.

The flexible spaces come in two flavors: interalign spaces and margin spaces. If there are n align structures ($n = 3$ in the illustration above), there are $n - 1$ interalign spaces, unless we are in an `alignat` environment, in which case there are no flexible interalign spaces.

The number of margin spaces is a little more complicated: Normally, there are two, but if we're in `fleqn` mode, there is only one. Furthermore, if we're in an `xxalignat` or `flalign` environment (corresponding to `\xatlevel@` = 2, then there are no flexible margin spaces.

Calculating the interalign and margin spaces is done in two stages.

First, the total amount of free space is divided uniformly among all the flexible spaces, without regard for the lengths of the tags on the various lines. For the non-`fleqn` case, this corresponds to centering the align structures between the margins. Note that in `fleqn` mode, the right margin is still allowed to be larger than `\@mathmargin`. This introduces an element of asymmetry into the appearance of the environment, but it has the advantage of leaving more space for equation tags in the right margin. If the right margin were constrained to be equal to the left margin in this case, tags would need to be shifted to a separate line more often than would be desirable.

Ordinarily, all flexible spaces will be given the same width. However, this is not invariably true, since the interalign spaces are constrained to be at least `\minalignsep` wide, while—in the absence of equation tags, at least—the margin spaces are allowed to shrink to zero. As we shall see in a minute, if there are tags in the environment, then the margins are also bounded below by `\mintagsep`.

Next, we examine each line of the environment that has a tag to see if there is a gap of at least `\mintagsep` between the equation and its tag. If there isn't, we attempt to center the equation between the tag and the opposite margin,

leaving a gap of at least \mintagsep on either side, in order to preserve some symmetry, i.e., we want the equation to *look* like it's centered between the margin and the tag, so we don't want the margin space to be less than the gap between the tag and the equation. (Arguably, it would be better to allow the margin space to shrink to zero in this case in order to avoid shifting the tag to a separate line at any cost, but that would require all of our calculations to be a little more complicated and hence a little slower.) Finally, if no values of the interalign spaces and the margins (with the constraints outlined above) will produce an acceptable distance between the equation and its tag, then the tag will be shifted to a separate line.

\measure@ \measure@ collects the various bits of information that we'll need to perform the calculations outlined above, namely, the number of align structures in the environment, the natural lengths of the fields on each row, the maximum widths of each column, and the widths of the equation tags on each line. It also calculates the number of flexible interalign and margin spaces and computes the initial values of the parameters \eqnshift@ and \alignsep@, which correspond to the widths of the margins and the interalign spaces, respectively.

```
\def\measure@#1{%
    \begingroup
        \measuring@true
        \global\eqnshift@\z@
        \global\alignsep@\z@
        \global\let\tag@lengths\@empty
        \global\let\field@lengths\@empty
        \savecounters@
        \global\setbox0\vbox{%
            \let\math@cr@@@\math@cr@@@align@measure
            \everycr{\noalign{\global\tag@false
              \global\let\raise@tag\@empty \global\column@\z@}}%
            \let\label\@gobble
            \global\row@\z@
            \tabskip\z@
            \halign{\span\align@preamble\crcr
                #1%
                \math@cr@@@
                \global\column@\z@
                \add@amps\maxfields@\cr
            }%
        }%
        \restorecounters@
```

It's convenient to have \maxfields@ rounded up to the nearest even number, so that \maxfields@ is precisely twice the number of align structures.

```
        \ifodd\maxfields@
            \global\advance\maxfields@\@ne
        \fi
```

It doesn't make sense to have a single align structure in either flalign or xxalignat. So, we check for that case now and, if necessary, switch to an align or alignat. Arguably, we should issue a warning message, but why bother?

```
        \ifnum\xatlevel@=\tw@
            \ifnum\maxfields@<\thr@@
                \let\xatlevel@\z@
            \fi
        \fi
```

\box0 now contains the lines of the \halign. After the following maneuver, \box1 will contain the last line of the \halign, which is what we're interested in. (Incidentally, the penalty we're removing is the \@eqpen inserted by \math@cr. Normally, this is \interdisplaylinepenalty, unless the user has overridden that with a \displaybreak command.)

```
\setbox\z@\vbox{%
  \unvbox\z@ \unpenalty \global\setbox\@ne\lastbox
}%
```

\box1 begins with \tabskip glue and contains alternating \hboxes (the fields whose widths we're trying to get) and \tabskip glue [need better diagram]:

```
\hbox{\tabskip\hbox\tabskip...\hbox\tabskip}
```

In fact, all the \tabskip glue will be 0pt, because all the \tabskips in an alignat environment have a natural width of 0pt, and the \halign has been set in its natural width.

One nice result of this is that we can read \totwidth@ off immediately, since it is just the width of \box1, plus \@mathmargin if we're in fleqn mode. (Actually, we also have to take \minalignsep into account, but we'll do that later):

```
\global\totwidth@\wd\@ne
\if@fleqn \global\advance\totwidth@\@mathmargin \fi
```

Now we initialize \align@lengths and start peeling the boxes off, one by one, and adding their widths to \align@lengths. We stop when we run out of boxes, i.e., when \lastbox returns a void box. We're going to build a list using \or as a delimiter, so we want to disable it temporarily.

```
\global\let\maxcolumn@widths\@empty
\begingroup
  \let\or\relax
  \loop
    \global\setbox\@ne\hbox{%
      \unhbox\@ne \unskip \global\setbox\thr@@\lastbox
    }%
  \ifhbox\thr@@
   \xdef\maxcolumn@widths{ \or \the\wd\thr@@ \maxcolumn@widths}%
  \repeat
\endgroup
```

Now we calculate the number of flexible spaces and the initial values of \eqnshift@ and \alignsep@. We start by calculating \displaywidth − \totwidth@, which gives us the total amount of "free space" in a row.

```
\dimen@\displaywidth
\advance\dimen@-\totwidth@
```

Next we calculate the number of columns of flexible spaces in the display, which depends on whether we're in fleqn mode and in which particular environment we are in.

We use \@tempcnta to store the total number of flexible spaces in the align and \@tempcntb for the number of interalign spaces.

```
\ifcase\xatlevel@
```

In alignat, the interalign spaces are under user control, not ours. So, we set \alignsep@ and \minalignsep both equal to 0pt. Later, when calculating a new value for \alignsep@, we will only save the new value if it is less than the current value of \alignsep@ (i.e., \alignsep@ will never increase). Since the values we calculate will never be negative, this will ensure that \alignsep@ remains zero in alignat.

```
            \global\alignsep@\z@
            \let\minalignsep\z@
            \@tempcntb\z@
```

In `fleqn` mode, the left margin—and hence the right margin in this case—is fixed. Otherwise, we divide the free space equally between the two margins.

```
            \if@fleqn
                \@tempcnta\@ne
                \global\eqnshift@\@mathmargin
            \else
                \@tempcnta\tw@
                \global\eqnshift@\dimen@
                \global\divide\eqnshift@\@tempcnta
            \fi
        \or
```

In an `align` or `xalignat` environment with n aligned structures, there are $n-1$ interalign spaces and either 1 or 2 flexible margins, depending on whether we're in `fleqn` mode or not.

```
            \@tempcntb\maxfields@
            \divide\@tempcntb\tw@
            \@tempcnta\@tempcntb
            \advance\@tempcntb\m@ne
```

If we are in `fleqn` mode, we fix the left margin and divide the free space equally among the interalign spaces and the right margin.

```
            \if@fleqn
                \global\eqnshift@\@mathmargin
                \global\alignsep@\dimen@
                \global\divide\alignsep@\@tempcnta
            \else
```

Otherwise, we divide the free space equally among the interalign spaces and both margins.

```
                \global\advance\@tempcnta\@ne
                \global\eqnshift@\dimen@
                \global\divide\eqnshift@\@tempcnta
                \global\alignsep@\eqnshift@
            \fi
        \or
```

Finally, if we're in an `flalign` or `xxalignat` environment, there are no flexible margins and $n-1$ flexible interalign spaces.

```
            \@tempcntb\maxfields@
            \divide\@tempcntb\tw@
            \global\advance\@tempcntb\m@ne
            \global\@tempcnta\@tempcntb
            \global\eqnshift@\z@
            \global\alignsep@\dimen@
```

If we're in `fleqn` mode, we need to add back the `\@mathmargin` that was removed when `\dimen@` was originally calculated above.

```
            \if@fleqn
                \global\advance\alignsep@\@mathmargin\relax
            \fi
            \global\divide\alignsep@\@tempcntb
        \fi
```

Now we make sure `\alignsep@` isn't too small.

```
        \ifdim\alignsep@<\minalignsep\relax
            \global\alignsep@\minalignsep\relax
```

```
            \ifdim\eqnshift@>\z@
                \if@fleqn\else
                    \global\eqnshift@\displaywidth
                    \global\advance\eqnshift@-\totwidth@
                    \global\advance\eqnshift@-\@tempcntb\alignsep@
                    \global\divide\eqnshift@\tw@
                \fi
            \fi
        \fi
        \ifdim\eqnshift@<\z@
            \global\eqnshift@\z@
        \fi
        \calc@shift@align
```

Next, we calculate the value of `\tagshift@`. This is the glue that will be inserted in front of the equation tag to make sure it lines up flush against the appropriate margin.

```
        \global\tagshift@\totwidth@
        \global\advance\tagshift@\@tempcntb\alignsep@
        \if@fleqn
            \ifnum\xatlevel@=\tw@
                \global\advance\tagshift@-\@mathmargin\relax
            \fi
        \else
            \global\advance\tagshift@\eqnshift@
        \fi
        \iftagsleft@ \else
            \global\advance\tagshift@-\displaywidth
        \fi
```

Finally, we increase `\totwidth@` by an appropriate multiple of `\minalignsep`. If the result is greater than `\displaywidth`, it means that at least one line in the `align` is overfull and we will issue an appropriate warning message (via `\bl@ck`) at the end of the environment.

```
        \dimen@\minalignsep\relax
        \global\advance\totwidth@\@tempcntb\dimen@
        \ifdim\totwidth@>\displaywidth
            \global\let\displaywidth@\totwidth@
        \else
            \global\let\displaywidth@\displaywidth
        \fi
    \endgroup
}
```

The code for calculating the appropriate placement of equation tags in the `align` environments is quite complicated and varies wildly depending on the settings of the `tagsleft@` and `@fleqn` switches. To minimize memory and hash space usage, we only define the variant appropriate for the current setting of those switches.

It would be worthwhile to examine this code more closely someday and see if it could be optimized any.

Tag placement when `\tagsleft@true`, `\@fleqntrue`. We begin with the version of `\calc@shift@align` appropriate for flush-left equations with tags on the left.

`\calc@shift@align` This is the simplest case. Since the left margin is fixed, in general the only thing to do is check whether there is room for the tag in the left margin. The only exception is that if `\eqnshift@` $= 0\,\mathrm{pt}$—meaning that we're in a `flalign`

environment and this is the first line with a tag that we've encountered—then we set `\eqnshift@` = `\@mathmargin` and recalculate `\alignsep@`. This is done by `\x@calc@shift@lf`.

```
\iftagsleft@\if@fleqn
    \def\calc@shift@align{%
        \global\let\tag@shifts\@empty
        \begingroup
```

`\@tempdima` is initialized to `\@mathmargin`−`\mintagsep`, which yields the maximum size of a tag that will not be shifted to another line.

```
            \@tempdima\@mathmargin\relax
            \advance\@tempdima-\mintagsep\relax
```

Now we examine each row in turn. If the width of the tag on the line is nonpositive—meaning either that there is no tag or else that the user has forced it to have zero width—we mark the tag to remain unshifted. Otherwise, we call `\x@calc@shift@lf` to determine whether any adjustments need to be made to `\eqnshift@` and `\alignsep@`. Note the difference in treatment of zero-width tags between this code and TEX's built-in algorithm: here, a width of zero prohibits the tag from being shifted, while in TEX's built-in algorithm, a width of zero forces the tag to be shifted.

```
            \loop
                \ifnum\row@>0
                    \ifdim\tag@width\row@>\z@
                        \x@calc@shift@lf
                    \else
                        \saveshift@0%
                    \fi
                    \advance\row@\m@ne
            \repeat
        \endgroup
    }
```

`\x@calc@shift@lf` As mentioned above, `\x@calc@shift@lf` first checks to see if the current left margin is set to 0 and, if so, resets it to `\@mathmargin` and recalculates `\alignsep@`. Next, it checks whether the length of the current tag exceeds the previously calculated limit and, if so, marks the tag to be shifted to a separate line.

```
    \def\x@calc@shift@lf{%
        \ifdim\eqnshift@=\z@
            \global\eqnshift@\@mathmargin\relax
            \alignsep@\displaywidth
            \advance\alignsep@-\totwidth@
            \global\divide\alignsep@\@tempcntb
            \ifdim\alignsep@<\minalignsep\relax
                \global\alignsep@\minalignsep\relax
            \fi
        \fi
        \ifdim\tag@width\row@>\@tempdima
            \saveshift@1%
        \else
            \saveshift@0%
        \fi
    }
\fi\fi
```

Tag placement when `\tagsleft@false`, `\@fleqntrue`. Next we consider the case when equations are flush-left, but tags are on the right. This case

is somewhat more complicated than the previous one, since we can adjust the right margin by varying the inter-align separatin. Thus, when a tag is found to be too close to its equation, we first attempt to decrease `\alignsep@` enough to move the equation off to an acceptable distance. Only if that would require a value of `\alignsep@` less than `\minalignsep` do we move the tag to a separate line.

`\calc@shift@align` This version of `\calc@shift@align` differs from the previous version only in calling `\x@calc@shift@rf` rather than `\x@calc@shift@lf`.

```
\iftagsleft@\else\if@fleqn
    \def\calc@shift@align{%
        \global\let\tag@shifts\@empty
        \begingroup
            \loop
                \ifnum\row@>0
                    \ifdim\tag@width\row@>\z@
                        \x@calc@shift@rf
                    \else
                        \saveshift@0%
                    \fi
                    \advance\row@\m@ne
            \repeat
        \endgroup
    }
```

`\x@calc@shift@rf` To start, we need to know two quantities: the number of align structures in the current row and the "effective length" of the row, defined as the distance from the left margin to the right edge of the text assuming that `\eqnshift@` and `\alignsep@` are both 0. To get the number of align structures, we first count the number of columns by counting the number of entries in the `\fieldlengths@` for the current row. The effective length is calcuated by `\x@rcalc@width` and put in the temporary register `\@tempdimc`, using `\@tempdimb` as an auxiliary variable.

```
    \def\x@calc@shift@rf{%
        \column@\z@
        \@tempdimb\z@
        \@tempdimc\z@
        \edef\@tempb{\fieldlengths@\row@}%
        \@for\@tempa:=\@tempb\do{%
            \advance\column@\@ne
            \x@rcalc@width
        }%
        \begingroup
```

If there are n columns in the current row, then there are $\lfloor (n+1)/2 \rfloor$ align structures and $\lfloor (n-1)/2 \rfloor$ interalign spaces.

```
            \advance\column@\m@ne
            \divide\column@\tw@
```

If this is smaller than the maximum number of interalign spaces in the environment, then we need to reduce `\@tempcnta` (the total number of flexible spaces in the current line) by `\@tempcntb`$-$`\column@` and reset `\@tempcntb` to `\column@`.

```
            \ifnum\@tempcntb>\column@
                \advance\@tempcnta-\@tempcntb
                \advance\@tempcnta\column@
                \@tempcntb\column@
            \fi
```

Next, we add the width of the tag and the (fixed) left margin to the effective length calculated above. This can be used to calculate how much "free space" there is in the current line and thus how much leeway we have to increase the amount of space between the tag and the equation.

```
            \tagwidth@\tag@width\row@\relax
            \@tempdima\eqnshift@
            \advance\@tempdima\@tempdimc\relax
            \advance\@tempdima\tagwidth@
```

The first thing to check is whether the tag should be shifted to a separate line. To do this, we add the minimum interalign separation and the `\mintagsep` to the value of `\@tempdima` just calculated. This yields the minimum acceptable length of the current line. If that is greater than `\displaywidth`, we mark the tag to be calculated. Otherwise, we mark the tag to be kept on the same line and then check to see if the `\alignsep@` needs to be reduced to make room for the tag.

```
            \dimen@\minalignsep\relax
            \multiply\dimen@\@tempcntb
            \advance\dimen@\mintagsep\relax
            \advance\dimen@\@tempdima
            \ifdim\dimen@>\displaywidth
                \saveshift@1%
            \else
                \saveshift@0%
```

Now we perform essentially the same calculation, but using the current value of `\alignsep@` instead of `\minalignsep`. This gives the current length of the line. If this is greater than `\displaywidth`, we recalculate `\alignsep@` to make room for the tag.

```
                \dimen@\alignsep@\relax
                \multiply\dimen@\@tempcntb
                \advance\dimen@\@tempdima
                \advance\dimen@\tagwidth@
                \ifdim\dimen@>\displaywidth
                    \dimen@\displaywidth
                    \advance\dimen@-\@tempdima
                    \ifnum\xatlevel@=\tw@
                        \advance\dimen@-\mintagsep\relax
                    \fi
                    \divide\dimen@\@tempcnta
                    \ifdim\dimen@<\minalignsep\relax
                        \global\alignsep@\minalignsep\relax
                    \else
                        \global\alignsep@\dimen@
                    \fi
                \fi
            \fi
        \endgroup
    }
\fi\fi
```

Tag placement when `\tagsleft@false`, `\@fleqnfalse`. This is similar to the previous case, except for the added complication that both `\alignsep@` and `\eqnshift@` can vary, which makes the computations correspondingly more complicated.

`\calc@shift@align`

```
\iftagsleft@\else\if@fleqn\else
```

```
\def\calc@shift@align{%
    \global\let\tag@shifts\@empty
    \begingroup
        \loop
            \ifnum\row@>0
                \ifdim\tag@width\row@>\z@
                    \x@calc@shift@rc
                \else
                    \saveshift@0%
                \fi
                \advance\row@\m@ne
        \repeat
    \endgroup
}
```

`\x@calc@shift@rc`

```
\def\x@calc@shift@rc{%
    \column@\z@
    \@tempdimb\z@
    \@tempdimc\z@
    \edef\@tempb{\fieldlengths@\row@}%
    \@for\@tempa:=\@tempb\do{%
        \advance\column@\@ne
        \x@rcalc@width
    }%
    \begingroup
        \advance\column@\m@ne
        \divide\column@\tw@
        \ifnum\@tempcntb>\column@
            \advance\@tempcnta-\@tempcntb
            \advance\@tempcnta\column@
            \@tempcntb\column@
        \fi
        \tagwidth@\tag@width\row@\relax
        \@tempdima\@tempdimc
        \advance\@tempdima\tagwidth@
        \dimen@\minalignsep\relax
        \multiply\dimen@\@tempcntb
        \advance\dimen@\mintagsep\relax
        \ifnum\xatlevel@=\tw@ \else
            \advance\dimen@\mintagsep\relax
        \fi
        \advance\dimen@\@tempdima
        \ifdim\dimen@>\displaywidth
            \saveshift@1%
        \else
            \saveshift@0%
            \dimen@\eqnshift@
            \advance\dimen@\@tempdima
            \advance\dimen@\@tempcntb\alignsep@
            \advance\dimen@\tagwidth@
            \ifdim\dimen@>\displaywidth
                \dimen@\displaywidth
                \advance\dimen@-\@tempdima
                \ifnum\xatlevel@=\tw@
                    \advance\dimen@-\mintagsep\relax
                \fi
                \divide\dimen@\@tempcnta
                \ifdim\dimen@<\minalignsep\relax
```

```
                        \global\alignsep@\minalignsep\relax
                        \eqnshift@\displaywidth
                        \advance\eqnshift@-\@tempdima
                        \advance\eqnshift@-\@tempcntb\alignsep@
                        \global\divide\eqnshift@\tw@
                    \else
                        \ifdim\dimen@<\eqnshift@
                            \ifdim\dimen@<\z@
                                \global\eqnshift@\z@
                            \else
                                \global\eqnshift@\dimen@
                            \fi
                        \fi
                        \ifdim\dimen@<\alignsep@
                            \global\alignsep@\dimen@
                        \fi
                    \fi
                \fi
            \fi
        \endgroup
    }
\fi\fi
```

`\x@rcalc@width`

```
\iftagsleft@\else
    \def\x@rcalc@width{%
        \ifdim\@tempa > \z@
            \advance\@tempdimc\@tempdimb
            \ifodd\column@
                \advance\@tempdimc\maxcol@width\column@
                \@tempdimb\z@
            \else
                \advance\@tempdimc\@tempa\relax
                \@tempdimb\maxcol@width\column@
                \advance\@tempdimb-\@tempa\relax
            \fi
        \else
            \advance\@tempdimb\maxcol@width\column@\relax
        \fi
    }
\fi
```

Tag placement when `\tagsleft@true`, `\@fleqnfalse`.

`\calc@shift@align`

```
\iftagsleft@\if@fleqn\else
    \def\calc@shift@align{%
        \global\let\tag@shifts\@empty
        \begingroup
            \loop
                \ifnum\row@>\z@
                    \ifdim\tag@width\row@>\z@
                        \x@calc@shift@lc
                    \else
                        \saveshift@0%
                    \fi
                    \advance\row@\m@ne
            \repeat
        \endgroup
```

```
1932    }
```

`\x@calc@shift@lc`

```
1933    \def\x@calc@shift@lc{%
1934        \column@\z@
```

`\@tempdima` will (eventually) be set to the effective width of the current row, defined as the distance from the leftmost point of the current line to the end of the last field of the `\halign`, ignoring any intervening `\tabskip`s, plus the width of the current tag. That is, it will be the width of the first non-empty field plus the sum of the maximum widths of all following fields, plus the tag width.

`\@tempdimb` will be the "indentation" of leftmost end of text, ignoring the `\tabskip` glue, i.e., it will be the sum of the maximum widths of any fields to the left of the first non-empty field, plus whatever empty space there is at the beginning of the first non-empty field.

```
1935        \@tempdima\z@ % ``width of equation''
1936        \@tempdimb\z@ % ``indent of equation''
1937        \edef\@tempb{\fieldlengths@\row@}%
1938        \@for\@tempa:=\@tempb\do{%
1939            \advance\column@\@ne
1940            \x@lcalc@width
1941        }%
1942        \begingroup
1943            \tagwidth@\tag@width\row@\relax
```

`\@tempdima` is now easy to calculate, since it is just `\totwidth@`$-$`\@tempdimb`$+$ `\tagwidth@`.

```
1944            \@tempdima\totwidth@
1945            \advance\@tempdima-\@tempdimb
1946            \advance\@tempdima\tagwidth@
```

Next, we check to see whether there is room for both the equation and the tag on the same line, by calculating the minimum acceptable length of the current row and comparing that to `\displaywidth`. Note that here we use `\@tempcntb`, i.e., the number of interalign spaces after the first non-empty align structure.

```
1947            \dimen@\minalignsep\relax
1948            \multiply\dimen@\@tempcntb
1949            \advance\dimen@\mintagsep\relax
1950            \ifnum\xatlevel@=\tw@ \else
1951                \advance\dimen@\mintagsep\relax
1952            \fi
1953            \advance\dimen@\@tempdima
```

If the minimum acceptable width of the current line is greater than `\displaywidth`, we mark the current tag to be shifted to a separate line.

```
1954            \ifdim\dimen@>\displaywidth
1955                \saveshift@1%
1956            \else
```

Otherwise, the tag can stay on the same line as the equation, but we need to check whether it is too close to the equation. So, we calculate the distance between the left margin and the left side of the equation, using the current values of `\eqnshift@` and `\alignsep@`. Note that we use `\count@` here, not `\@tempcntb`, as above.

```
1957                \saveshift@0%
1958                \dimen@\alignsep@
1959                \multiply\dimen@\count@
1960                \advance\dimen@\eqnshift@
1961                \advance\dimen@\@tempdimb
```

If the left margin is less than twice the tag width, we calculate new values of `\eqnshift@` and `\alignsep@` to move the equation further away from the tag. In particular, we center the current line between its tag and the right margin. Note that although we later will need to transform `\dimen@` into a value suitable for use as `\eqnshift@`, for the time being it is more useful to think of it as the space separating the tag from the equation.

```
\ifdim\dimen@<2\tagwidth@
    \dimen@\displaywidth
    \advance\dimen@-\@tempdima
    \ifnum\xatlevel@=\tw@
        \advance\dimen@-\mintagsep\relax
    \fi
```

In certain circumstances we will get a divide-by-zero error here unless we guard against it. Use of `\@tempcnta` is complicated, sometimes it is assigned globally, sometimes locally. Need to sort it out one of these days [mjd,2000/06/02].

```
\ifnum\@tempcnta>\z@
  \divide\dimen@\@tempcnta
\else \dimen@\z@
\fi
```

As usual, we check to make sure we don't set `\alignsep@` smaller than `\minalignsep` and, in any case, that we don't replace `\alignsep@` by a larger value.

```
\ifdim\dimen@<\minalignsep\relax
    \global\alignsep@\minalignsep\relax
    \dimen@\displaywidth
    \advance\dimen@-\@tempdima
    \advance\dimen@-\@tempcntb\alignsep@
    \global\divide\dimen@\tw@
\else
    \ifdim\dimen@<\alignsep@
        \global\alignsep@\dimen@
    \fi
\fi
```

Next, we calculate an appropriate value of `\eqnshift@`, assuming that `\dimen@` is the desired separation between the tag and equation of the current line. This means that we first need to adjust `\dimen@` if we're in an `flalign` environment.

```
\ifnum\xatlevel@=\tw@
    \dimen@\mintagsep\relax
\fi
```

Now we calculate the value of `\eqnshift@` needed to produce a separation of `\dimen@` between the equation tag and the beginning of the equation. To do this, we need the following equation to hold:

$$\texttt{\eqnshift@} + n\texttt{\alignsep@} + \texttt{\@tempdimb} = \texttt{\tagwidth@} + \texttt{\dimen@}$$

where $n = \texttt{\count@}$ is the number of interalign spaces before the first non-empty field of the current line.

```
\advance\dimen@\tagwidth@
\advance\dimen@-\@tempdimb
\advance\dimen@-\count@\alignsep@
```

The value of `\eqnshift@` just calculated is the minimum acceptable value; thus, we save it only if it is larger than the current value.

```
\ifdim\dimen@>\eqnshift@
    \global\eqnshift@\dimen@
\fi
```

```
                \fi
            \fi
        \endgroup
    }
```

`\x@lcalc@width` This macro calculates the "indentation" of the current row, as defined above under the description of `\x@calc@shift@lc`. This macro is called for each field of the current line, with `\@tempa` set to the width of the current field. Ideally, the loop enclosing `\x@lcalc@width` would terminate as soon as `\@tempa` is non-zero, but that would be a bit tricky to arrange. Instead, we use `\@tempdima` as a flag to signal when we've encountered the first non-empty field.

```
    \def\x@lcalc@width{%
        \ifdim\@tempdima = \z@
```

If the current field is empty (i.e., `\@tempa` $= 0\,\mathrm{pt}$, then we increment `\@tempdimb` by the width of the current field). Otherwise, we set `\@tempdima` $= 1\,\mathrm{pt}$ as a signal value and increment `\@tempdimb` by the width of whatever empty space there might be at the left of the current field.

```
            \ifdim\@tempa > \z@
                \@tempdima\p@
                \ifodd\column@
                    \advance\@tempdimb \maxcol@width\column@
                    \advance\@tempdimb-\@tempa
                \fi
```

In addition, we need to adjust the values of `\@tempcnta` and `\@tempcntb` to account for any empty align structures that might occur at the beginning of the current line. More specifically, we first set `\count@` equal to the number of interalign spaces preceding the current field (namely, $\lfloor(\texttt{\column@} - 1)/2\rfloor$), and then subtract `\count@` from both `\@tempcnta` and `\@tempcntb`. The rationale is that for the purposes of adjusting the spacing between the tag and the equation, the only flexible interalign spaces are those after the first non-empty align structure, so we need to treat those different from the ones before the first non-empty align structure.

```
                \count@\column@
                \advance\count@\m@ne
                \divide\count@\tw@
                \advance\@tempcnta-\count@
                \advance\@tempcntb-\count@
            \else
                \advance\@tempdimb \maxcol@width\column@\relax
            \fi
        \fi
    }
\fi\fi
```

`\place@tag` `\place@tag` takes care of the placment of tags in the `align` environments.

```
\def\place@tag{%
    \iftagsleft@
        \kern-\tagshift@
        \if1\shift@tag\row@\relax
            \rlap{\vbox{%
                \normalbaselines
                \boxz@
                \vbox to\lineht@{}%
                \raise@tag
            }}%
        \else
```

```
            \rlap{\boxz@}%
        \fi
        \kern\displaywidth@
    \else
        \kern-\tagshift@
        \if1\shift@tag\row@\relax
```

Added depth to correct vertical spacing of shifted equation tags.—dmj, 1994/12/29

```
            \llap{\vtop{%
                \raise@tag
                \normalbaselines
                \setbox\@ne\null
                \dp\@ne\lineht@
                \box\@ne
                \boxz@
            }}%
        \else
            \llap{\boxz@}%
        \fi
    \fi
}
```

`\align@preamble`

```
\def\align@preamble{%
   &\hfil
    \strut@
    \setboxz@h{\@lign$\m@th\displaystyle{##}$}%
    \ifmeasuring@\savefieldlength@\fi
    \set@field
    \tabskip\z@skip
   &\setboxz@h{\@lign$\m@th\displaystyle{{}##}$}%
    \ifmeasuring@\savefieldlength@\fi
    \set@field
    \hfil
    \tabskip\alignsep@
}
```

`\set@field` `\set@field` increments the column counter, tracks the value of `\lineht@` and finally inserts the box containing the contents of the current field.

```
\def\set@field{%
    \column@plus
    \iftagsleft@
        \ifdim\ht\z@>\lineht@
            \global\lineht@\ht\z@
        \fi
    \else
        \ifdim\dp\z@>\lineht@
            \global\lineht@\dp\z@
        \fi
    \fi
    \boxz@
}
```

16.7 The `split` environment

`\split@err` A special error function for `split` to conserve main mem (at a cost of string pool/hash size.

```
\edef\split@err#1{%
    \@nx\@amsmath@err{%
        \string\begin{split} won't work here%
```

```
    }{%
        \@xp\@nx\csname
  Did you forget a preceding \string\begin{equation}?^^J%
  If not, perhaps the `aligned' environment is what
  you want.\endcsname}%
}
```

split If the split environment occurs inside align or gather, it can make use of the enclosing halign; if it is called inside a simple equation, we add an implicit 'gather' container.

```
\newenvironment{split}{%
  \if@display
    \ifinner
      \@xp\@xp\@xp\split@aligned
    \else
      \ifst@rred \else \global\@eqnswtrue \fi
    \fi
  \else \let\endsplit\@empty \@xp\collect@body\@xp\split@err
  \fi
  \collect@body\gather@split
}{%
      \crcr
    \egroup
  \egroup
  \iftagsleft@ \@xp\lendsplit@ \else \@xp\rendsplit@ \fi
}
```

```
\let\split@tag\relax % init
```

```
\def\gather@split#1#2#3{%
  \@xp\endgroup \reset@equation % math@cr will handle equation numbering
  \iftag@
     \toks@\@xp{\df@tag}%
     \edef\split@tag{%
       \gdef\@nx\df@tag{\the\toks@}%
       \global\@nx\tag@true \@nx\nonumber
     }%
  \else \let\split@tag\@empty
  \fi
  \spread@equation
```

The extra vcenter wrapper here is not really a good thing but without it there are compatibility problems with old documents that throw in some extra material between \begin{equation} and \begin{split} (for example, \hspace{-1pc} or \left\{). [mjd,1999/09/20]

```
  \vcenter\bgroup
    \gather@{\split@tag  \begin{split}#1\end{split}}%
    \def\endmathdisplay@a{%
      \math@cr \black@ \totwidth@ \egroup
      \egroup
    }%
}
```

\insplit@

```
\def\insplit@{%
  \global\setbox\z@\vbox\bgroup
    \Let@ \chardef\dspbrk@context\@ne \restore@math@cr
    \default@tag % disallow use of \tag here
    \ialign\bgroup
      \hfil
```

```
      \strut@
      $\m@th\displaystyle{##}$%
     &$\m@th\displaystyle{{}##}$%
      \hfill % Why not \hfil?---dmj, 1994/12/28
      \crcr
}
```

`\rendsplit@` Moved the box maneuvers inside the `\ifinalign@`, since that is the only place they are needed.—dmj, 1994/12/28

TODO: Explore interaction of tag-placement algorithm with `split`. Is there any way for `split` to pass the relevant information out to the enclosing `gather` or `align`?

```
\def\rendsplit@{%
    \ifinalign@
```

Changed `\box9` into a `\vtop` here for better spacing.

```
        \global\setbox9 \vtop{%
            \unvcopy\z@
            \global\setbox8 \lastbox
            \unskip
        }%
        \setbox\@ne\hbox{%
            \unhcopy8
            \unskip
            \global\setbox\tw@\lastbox
            \unskip
            \global\setbox\thr@@\lastbox
        }%
        \ifctagsplit@
            \gdef\split@{%
                \hbox to\wd\thr@@{}%
               &\vcenter{\vbox{\moveleft\wd\thr@@\boxz@}}%
            }%
        \else
            \global\setbox7 \hbox{\unhbox\tw@\unskip}%
```

Added `\add@amps` to make sure we put the last line of the `split` into the proper column of an `align` environment with multiple align structures.—dmj, 1994/12/28

Special care has to be taken in this case because the `split` turns into two lines of the `align` instead of just one. So, we have to make sure that the first line produced by the `split` doesn't upset our bookkeeping, hence we call `\savetaglength@` to insert 0 pt as the tag for this pseudo-line, and we advance the `\row@` counter and reset `\lineht@` afterwards. It would be nice if we could just replace the `\crcr` by `\math@cr@@@`, but that would cause problems with the tag processing.

```
            \gdef\split@{%
                \global\@tempcnta\column@
               &\setboxz@h{}%
                \savetaglength@
                \global\advance\row@\@ne
                \vbox{\moveleft\wd\thr@@\box9}%
                \crcr
                \noalign{\global\lineht@\z@}%
                \add@amps\@tempcnta
                \box\thr@@
               &\box7
            }%
        \fi
```

```
    \else
        \ifctagsplit@
            \gdef\split@{\vcenter{\boxz@}}%
        \else
```

Changed to just \boxz@, otherwise last line gets centered rather than aligned properly with respect to the rest of the lines. But this means that we can't see inside of the last line to decide whether the tag needs to be moved. Will have to think about this.—dmj, 1994/12/28

```
            \gdef\split@{%
                \boxz@
%                \box9
%                \crcr
%                \hbox{\box\thr@@\box7}%
            }%
        \fi
    \fi
    \aftergroup\split@
}
```

\lendsplit@

```
\def\lendsplit@{%
    \global\setbox9\vtop{\unvcopy\z@}%
    \ifinalign@
```

Moved following two boxes inside the \ifinalign@, since they are only used in that case. In fact, if we just kept track of the width of the first column, we could dispense with this entirely. Surely that would be more efficient than all these box copies.—dmj, 1994/12/28

```
        \setbox\@ne\vbox{%
            \unvcopy\z@
            \global\setbox8\lastbox
        }%
        \setbox\@ne\hbox{%
            \unhcopy8%
            \unskip
            \setbox\tw@\lastbox
            \unskip
            \global\setbox\thr@@\lastbox
        }%
        \ifctagsplit@
            \gdef\split@{%
                \hbox to\wd\thr@@{}%
               &\vcenter{\vbox{\moveleft\wd\thr@@\box9}}%
            }%
        \else
            \gdef\split@{%
                \hbox to\wd\thr@@{}%
               &\vbox{\moveleft\wd\thr@@\box9}%
            }%
        \fi
    \else
        \ifctagsplit@
            \gdef\split@{\vcenter{\box9}}%
        \else
            \gdef\split@{\box9}%
        \fi
    \fi
    \aftergroup\split@
}
```

With amsmath 1.2 it was possible to put things like \left\{ between \begin{equation} and \begin{split} without getting any error message. For backward compatibility we try to avoid a fatal error in this case and instead attempt recovery with aligned.

```
\def\split@aligned#1#2{%
   \iffalse{\fi\ifnum0=`}\fi
   \collect@body\split@al@a}

\def\split@al@a#1#2#3{%
  \split@warning
  \endgroup
```

If the fleqn and tbtags options are both in effect then we will need to add an optional argument on the aligned environment.

```
  \toks@{\begin{aligned}}%
  \if@fleqn \split@al@tagcheck \fi
```

The \relax here is to prevent \@let@token from being left equal to an ampersand if that happens to be the first thing in the body.

```
  \the\toks@\relax#1\end{aligned}%
  \ifnum0=`{\fi\iffalse}\fi
}

\def\split@al@tagcheck{%
   \ifctagsplit@
   \else
     \iftagsleft@ \toks@\@xp{\the\toks@ [t]}%
     \else \toks@\@xp{\the\toks@ [b]}%
     \fi
   \fi
}

\def\split@warning{%
  \PackageWarning{amsmath}{%
Cannot use `split' here;\MessageBreak trying to recover with `aligned'}%
}
```

16.8 The multline environment

In the original $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX, \multlinegap is a macro with an argument that resets an internal dimension (one with an @ character in its name). Here, to save control sequence names, we define \multlinegap to be the dimension itself and the documentation instructs users to use \setlength if they need to change it.

\multlinegap
\multlinetaggap

Changed \multlinegap and \multlinetaggap to skip registers. Also changed name to \multlinetaggap from \multlinetaggap@.

```
\newskip\multlinegap
\multlinegap10pt
\newskip\multlinetaggap
\multlinetaggap10pt
```

\start@multline

```
\def\start@multline#1{%
    \RIfM@
        \nomath@env
        \DN@{\@namedef{end\@currenvir}{}\@gobble}%
    \else
        $$%
        #1%
        \ifst@rred
```

```
            \nonumber
        \else
            \global\@eqnswtrue
        \fi
        \let\next@\multline@
    \fi
    \collect@body\next@
}
```

multline
multline*

```
\newenvironment{multline}{%
  \start@multline\st@rredfalse
}{%
  \iftagsleft@ \@xp\lendmultline@ \else \@xp\rendmultline@ \fi
  \ignorespacesafterend
}
\newenvironment{multline*}{\start@multline\st@rredtrue}{\endmultline}
```

\multline@

```
\def\multline@#1{%
    \Let@
```

For multline neither `\displ@y` no `\displ@y@` is quite right; we want to advance the row number and (I suppose?) the display-pagebreak level, but we only want to do tag-related stuff once before the first line, not repeat it for every line. (Recall that the arg of `\@display@init` goes into `\everycr`.)

```
    \@display@init{\global\advance\row@\@ne \global\dspbrk@lvl\m@ne}%
    \chardef\dspbrk@context\z@
    \restore@math@cr
```

The `multline` environment is somewhat unusual, in that `\tag` and `\label` are enabled only during the measuring phase and disabled during the production phase. Here we disable `\tag` and `\label`; `\mmeasure@` will re-enable them temporarily.

```
    \let\tag\tag@in@align
    \global\tag@false \global\let\raise@tag\@empty
    \mmeasure@{#1}%
    \let\tag\gobble@tag \let\label\@gobble
    \tabskip \if@fleqn \@mathmargin \else \z@skip \fi
    \totwidth@\displaywidth
    \if@fleqn
        \advance\totwidth@-\@mathmargin
    \fi
    \halign\bgroup
        \hbox to\totwidth@{%
```

In order to get the spacing of the last line right in fleqn mode, we need to play a little game here. Normally the stretchability of the `\hskip` here will be suppressed by the `\hfil` at the end of the template, except inside the last line, when that `\hfil` will be removed by the `\hfilneg` in `\lendmultline@`.

```
            \if@fleqn
                \hskip \@centering \relax
            \else
                \hfil
            \fi
            \strut@
            $\m@th\displaystyle{}##\endmultline@math
            \hfil
        }%
        \crcr
```

In `fleqn` mode, it's the `\tabskip` of `\@mathmargin` that needs to be removed in the first line, not the `\hfil` at the beginning of the template.

```
        \if@fleqn
            \hskip-\@mathmargin
            \def\multline@indent{\hskip\@mathmargin}% put it back
        \else
            \hfilneg
            \def\multline@indent{\hskip\multlinegap}%
        \fi
        \iftagsleft@
            \iftag@
                \begingroup
                    \ifshifttag@
                        \rlap{\vbox{%
                                \normalbaselines
                                \hbox{%
                                    \strut@
                                    \make@display@tag
                                }%
                                \vbox to\lineht@{}%
                                \raise@tag
                        }}%
```

If the equation tag doesn't fit on the same line with the first line of the display, we'll indent the first line by `\multlinegap`. This is a change from `amstex`, where the first line would have been flush against the left margin in this case. A corresponding change will be made in `\rendmultline@`.

```
                        \multline@indent
                    \else
                        \setbox\z@\hbox{\make@display@tag}%
                        \dimen@\@mathmargin \advance\dimen@-\wd\z@
                        \ifdim\dimen@<\multlinetaggap
                          \dimen@\multlinetaggap
                        \fi
                        \box\z@ \hskip\dimen@\relax
                    \fi
                \endgroup
            \else
                \multline@indent
            \fi
        \else
            \multline@indent
        \fi
    #1%
}
```

An extra level of indirection for the closing `$` in multline allows us to avoid getting an extra thinmuskip from a final mathpunct in the equation contents, when equation numbers are on the right. If we did not use this workaround, the sequence of elements for a final comma would be, e.g.,

```
... ,<hskip><box containing equation number>
```

which is equivalent to a sequence `<mathpunct><mathord>` as far as the automatic math spacing is concerned.

```
\def\endmultline@math{$}
```

`\lendmultline@` Bug fix: changed `\crcr` to `\math@cr` so that `\@eqpen` gets reset properly if `\displaybreak` is used on the penultimate line of an `align`.

```
\def\lendmultline@{%
        \hfilneg
        \hskip\multlinegap
        \math@cr
    \egroup
    $$%
}
```

\rendmultline@

```
\def\rendmultline@{%
    \iftag@
        $\let\endmultline@math\relax
            \ifshifttag@
                \hskip\multlinegap
```

Added depth to correct vertical spacing of shifted equation tags.—dmj, 1994/12/29

```
                \llap{\vtop{%
                    \raise@tag
                    \normalbaselines
                    \setbox\@ne\null
                    \dp\@ne\lineht@
                    \box\@ne
                    \hbox{\strut@\make@display@tag}%
                }}%
            \else
                \hskip\multlinetaggap
                \make@display@tag
            \fi
    \else
        \hskip\multlinegap
    \fi
    \hfilneg
```

Use \math@cr rather than just \crcr so that \@eqpen gets reset properly if \displaybreak is used.

```
        \math@cr
    \egroup$$%
}
```

\mmeasure@

```
\def\mmeasure@#1{%
    \begingroup
        \measuring@true
```

We use \begin/endgroup rather than {} in this definition of \label because the latter would create an extra (wasteful of main mem) null box in the current math list. [mjd, 1995/01/17]

```
        \def\label##1{%
          \begingroup\measuring@false\label@in@display{##1}\endgroup}%
        \def\math@cr@@@{\cr}%
        \let\shoveleft\@iden \let\shoveright\@iden
        \savecounters@
        \global\row@\z@
        \setbox\@ne\vbox{%
            \global\let\df@tag\@empty
            \halign{%
                \setboxz@h{\@lign$\m@th\displaystyle{}##$}%
                \iftagsleft@
                    \ifnum\row@=\@ne
                        \global\totwidth@\wdz@
```

```
                        \global\lineht@\ht\z@
                    \fi
                \else
                    \global\totwidth@\wdz@
                    \global\lineht@\dp\z@
                \fi
                \crcr
                #1%
                \crcr
            }%
        }%
        \ifx\df@tag\@empty\else\global\tag@true\fi
        \if@eqnsw\global\tag@true\fi
        \iftag@
            \setboxz@h{%
                \if@eqnsw
                    \stepcounter{equation}%
                    \tagform@\theequation
                \else
                    \df@tag
                \fi
            }%
            \global\tagwidth@\wdz@
            \dimen@\totwidth@
            \advance\dimen@\tagwidth@
            \advance\dimen@\multlinetaggap
            \iftagsleft@\else
                \if@fleqn
                    \advance\dimen@\@mathmargin
                \fi
            \fi
            \ifdim\dimen@>\displaywidth
                \global\shifttag@true
            \else
                \global\shifttag@false
            \fi
        \fi
        \restorecounters@
    \endgroup
}
```

\shoveleft
\shoveright

`\shoveleft` and `\shoveright` need to do slightly different things depending on whether tags are on the left or the right and whether we're in `fleqn` mode. For compactness of code, we make the appropriate decisions at "compile" time rather than at load time.

TODO: Investigate making `\shoveright` behave "properly"(?) if used on the first line of a `multline` and make `\shoveleft` behave properly if used on the last line of a `multline`. But in his `amstex.doc` Spivak indicates those commands should never be used on a first or last line. Perhaps better to leave the question open unless/until real-life examples turn up.

```
\iftagsleft@
    \def\shoveright#1{%
        #1%
        \hfilneg
        \hskip\multlinegap
    }
\else
    \def\shoveright#1{%
        #1%
```

```
        \hfilneg
        \iftag@
            \ifshifttag@
                \hskip\multlinegap
            \else
                \hskip\tagwidth@
                \hskip\multlinetaggap
            \fi
        \else
            \hskip\multlinegap
        \fi
    }
\fi

\if@fleqn
    \def\shoveleft#1{#1}%
\else
    \iftagsleft@
        \def\shoveleft#1{%
            \setboxz@h{$\m@th\displaystyle{}#1$}%
            \setbox\@ne\hbox{$\m@th\displaystyle#1$}%
            \hfilneg
            \iftag@
                \ifshifttag@
                    \hskip\multlinegap
                \else
                    \hskip\tagwidth@
                    \hskip\multlinetaggap
                \fi
            \else
                \hskip\multlinegap
            \fi
            \hskip.5\wd\@ne
            \hskip-.5\wdz@
            #1%
        }
    \else
        \def\shoveleft#1{%
            \setboxz@h{$\m@th\displaystyle{}#1$}%
            \setbox\@ne\hbox{$\m@th\displaystyle#1$}%
            \hfilneg
            \hskip\multlinegap
            \hskip.5\wd\@ne
            \hskip-.5\wdz@
            #1%
        }
    \fi
\fi
```

16.9 The `equation` environment

Rewritten from the ground up for version 2.0 to fix no-shrink and no-shortskips bugs [mjd,2000/01/06].

Standard LaTeX provides three environments for one-line equations: `\[ \]`, `equation`, and `displaymath`. We add `equation*` as a synonym for `displaymath`.

```
\@saveprimitive\leqno\@@leqno
\@saveprimitive\eqno\@@eqno
\def\eqno{\@@eqno\let\eqno\relax\let\leqno\relax}
```

```
\def\leqno{\@@leqno\let\leqno\relax\let\eqno\relax}
%
\let\veqno=\@@eqno
\iftagsleft@ \let\veqno=\@@leqno \fi
```

Support for the `showkeys` package: provide no-op definitions for a couple of SK functions, if they are not already defined. Then we can just call them directly in our code without any extra fuss. If the `showkeys` package is loaded later, our trivial definitions will get overridden and everything works fine.

```
\@ifundefined{SK@@label}{%
  \let\SK@@label\relax \let\SK@equationtrue\relax
}{}
\let\reset@equation\@empty
```

Cf `\tag@in@align`. This is a bit of a mess though. Could use some work. [mjd,1999/12/21]

```
\let\alt@tag\@empty
\def\tag@in@display#1#{\relax\tag@in@display@a{#1}}
\def\tag@in@display@a#1#2{%
  \iftag@
    \invalid@tag{Multiple \string\tag}\relax
  \else
    \global\tag@true \nonumber \reset@equation \st@rredtrue
    \if *\string#1%
      \gdef\alt@tag{\def\SK@tagform@{#2\@gobble}%
        \ifx\SK@@label\relax \let\tagform@\SK@tagform@ \fi
      }%
      \make@df@tag@@{#2}%
    \else
      \make@df@tag@@@{#2}%
    \fi
  \fi
}
\let\restore@hfuzz\@empty
\def\mathdisplay#1{%
  \ifmmode \@badmath
  \else
    $$\def\@currenvir{#1}%
```

Allow use of `\displaybreak`.

```
    \let\dspbrk@context\z@
```

Although in some cases simpler label handling would seem to be sufficient, always using `\label@in@display` makes it easier to support the `showkeys` package.

```
    \let\tag\tag@in@display \let\label\label@in@display \SK@equationtrue
    \global\let\df@label\@empty \global\let\df@tag\@empty
    \global\tag@false
    \let\mathdisplay@push\mathdisplay@@push
    \let\mathdisplay@pop\mathdisplay@@pop
    \if@fleqn
```

Turn off overfull box messages temporarily—otherwise there would be unwanted extra ones emitted during our measuring operations.

```
      \edef\restore@hfuzz{\hfuzz\the\hfuzz\relax}%
      \hfuzz\maxdimen
```

Initially set the equation body in a box of displaywidth. Then if the box is not overfull, as we find by checking `\badness`, we have acquired useful information for the subsequent processing.

```
      \setbox\z@\hbox to\displaywidth\bgroup
        \let\split@warning\relax \restore@hfuzz
        \everymath\@emptytoks \m@th $\displaystyle
    \fi
  \fi
}
```

Arg 1 is not currently used. I thought it might come in handy for error messages.

```
\def\endmathdisplay#1{%
  \ifmmode \else \@badmath \fi
  \endmathdisplay@a
  $$%
```

I guess the following code means this structure is non-reentrant. But there is plenty of scope for tricky bugs here; suppressing them by brute force at least makes it possible to get things working correctly for normal use. [mjd,2000/01/06]

```
  \global\let\df@label\@empty \global\let\df@tag\@empty
  \global\tag@false \global\let\alt@tag\@empty
  \global\@eqnswfalse
}
\def\endmathdisplay@a{%
  \if@eqnsw \gdef\df@tag{\tagform@\theequation}\fi
  \if@fleqn \@xp\endmathdisplay@fleqn
  \else \ifx\df@tag\@empty \else \veqno \alt@tag \df@tag \fi
    \ifx\df@label\@empty \else \@xp\ltx@label\@xp{\df@label}\fi
  \fi
  \ifnum\dspbrk@lvl>\m@ne
    \postdisplaypenalty -\@getpen\dspbrk@lvl
    \global\dspbrk@lvl\m@ne
  \fi
}
```

A boolean variable: Was that last box overfull or not? A value of 0 means yes, it was overfull.

```
\let\too@wide\@ne
```

Special handling is needed for flush-left equations. We need to measure the equation body (found in box 0 after we close it with the `\egroup`). Then after a fairly normal test to see if it fits within the available space, we need to consider overlapping into the displayindent area if displayindent is nonzero (as in an indented list). If there is an equation number we may have to shift it by hand to a separate line when there is not enough room; we can no longer take advantage of the automatic shifting provided by the `\leqno`, `\eqno` primitives.

We initially add `\@mathmargin` glue at the end of box 0 to get an accurate overfull test. If `\@mathmargin` contains any shrink then we cannot reliably tell whether the box will be overfull or not simply by doing hand calculations from the actual width of the equation body. We have to actually set the box and find out what happens.

On the other hand if we put the `\@mathmargin` glue at the beginning of the box it's awkward to remove it afterwards. So we first put it in at the end and later we will move it to the beginning as needed.

```
\def\endmathdisplay@fleqn{%
  $\hfil\hskip\@mathmargin\egroup
```

We need to save the information about whether box 0 was overfull in a variable, otherwise it will disappear in the next setbox operation. And we couldn't set the equation number box earlier than now, because the body of the equation

might have contained a \tag command (well, it could have been done, but this way we can reuse the tag-handling code from elsewhere).

```
  \ifnum\badness<\inf@bad \let\too@wide\@ne \else \let\too@wide\z@ \fi
  \ifx\@empty\df@tag
  \else
    \setbox4\hbox{\df@tag
      \ifx\df@label\@empty \else \@xp\ltx@label\@xp{\df@label}\fi
    }%
  \fi
  \csname emdf@%
    \ifx\df@tag\@empty U\else \iftagsleft@ L\else R\fi\fi
  \endcsname
}
```

For an unnumbered flush-left equation we hope first that the the contents fit within displaywidth. If not we need to fall back on a more complicated reboxing operation.

```
\def\emdf@U{%
  \restore@hfuzz
  \ifodd\too@wide % not too wide: just need to swap the glue around
    \hbox to\displaywidth{\hskip\@mathmargin\unhbox\z@\unskip}%
  \else % M+B > displaywidth
    \emdf@Ua
  \fi
}
```

Some notation: M \@mathmargin, B the width of the equation body, I \displayindent, D \displaywidth, N the width of the equation number (aka the tag), S \mintagsep, C \columnwidth. If $M + B >$ displaywidth, and if we assume M contains shrink, then the only solution left is to encroach into the displayindent space.

```
\def\emdf@Ua{%
  \hbox to\columnwidth{%
    \ifdim\displayindent>\z@
      \hskip\displayindent minus\displayindent
    \fi
    \hskip\@mathmargin \unhbox\z@ \unskip
  }%
  \displayindent\z@ \displaywidth\columnwidth
}
```

Find out first if the tag fits in ideal position. If so we can just plunk down box 2. Otherwise we need to do something more complicated.

```
\def\emdf@R{%
  \setbox\tw@\hbox to\displaywidth{%
    \hskip\@mathmargin \unhcopy\z@\unskip\hfil\hskip\mintagsep\copy4
  }%
  \restore@hfuzz
  \ifnum\badness<\inf@bad \box\tw@ \else \emdf@Ra \fi
}
```

We shift the equation number to line 2 if it does not fit within \displaywidth. Note that we do not first attempt to let the equation body shift leftward into the \displayindent space. If that is desired it will have to be done by hand by adding negative space at the beginning of the equation body. I don't expect this to arise very often in practice since most of the time \displayindent is zero anyway.

```
\def\emdf@Ra{%
  \skip@\displayindent minus\displayindent
```

```
2575   \displayindent\z@ \displaywidth\columnwidth
2576   \spread@equation \everycr{}\tabskip\z@skip
2577   \halign{\hbox to\displaywidth{##}\cr
2578     \relax
2579     \ifdim\skip@>\z@ \hskip\skip@ \fi
2580     \hskip\@mathmargin\unhbox\z@\unskip\hfil\cr
2581     \noalign{\raise@tag}%
2582     \hfil\box4 \cr}%
2583 }
```

Find out first if the tag fits in ideal position. If so we can just plunk down box 2. Otherwise we need to do something more complicated.

```
2584 \def\emdf@L{%
```

Calculate the difference between M and $N + S$. If the latter is greater, we don't want to add any extra glue between the number and the equation body. Otherwise the amount that we want to add is `x minus x` where $x = M-(N+S)$. I.e., the distribution of spaces across the line is $N, S, xminusx, B, hfil$.

```
2585   \@tempdima\@mathmargin
2586   \advance\@tempdima-\wd4 \advance\@tempdima-\mintagsep
2587   \skip@\@tempdima minus\@tempdima
2588   \setbox\tw@\hbox to\displaywidth{%
2589     \copy4\hskip\mintagsep
2590     \ifdim\skip@>\z@ \hskip\skip@\fi
2591     \unhcopy\z@\unskip
2592   }%
2593   \restore@hfuzz
2594   \ifnum\badness<\inf@bad \box\tw@ \else \emdf@La \fi
2595 }
```

If the equation body and equation number will not fit on the same line, we put the number on line 1 and the body on line 2, with the body positioned as for an unnumbered equation.

```
2596 \def\emdf@La{%
2597   \spread@equation \everycr{}\tabskip\z@skip
2598   \halign{\hbox to\displaywidth{##}\cr
2599     \box4 \hfil \cr
2600     \noalign{\raise@tag}%
2601     \hskip\@mathmargin\unhbox\z@\unskip\hfil\cr}%
2602 }
```

If someone has `\[ \]` nested inside a minipage environment nested inside a numbered equation, the mathdisplay variables that are global will get out of whack unless we take extra care. So we make a stack and push all the variables before entering mathdisplay and pop them afterwards. But we can save a little work by not doing this at the top level, only at inner levels.

```
2603 \newtoks\mathdisplay@stack
2604 \let\mathdisplay@push\@empty
2605 \def\mathdisplay@@push{%
2606   \begingroup
2607   \toks@\@xp{\df@label}\@temptokena\@xp{\df@tag}%
2608   \toks8\@xp{\alt@tag}%
2609   \edef\@tempa{%
2610     \global\if@eqnsw\@nx\@eqnswtrue\else\@nx\@eqnswfalse\fi
2611     \global\iftag@\@nx\tag@false\else\@nx\tag@true\fi
2612     \gdef\@nx\df@label{\the\toks@}\gdef\@nx\df@tag{\the\@temptokena}%
2613     \gdef\@nx\alt@tag{\the\toks8}%
2614     \global\mathdisplay@stack{\the\mathdisplay@stack}%
2615   }%
2616   \global\mathdisplay@stack\@xp{\@tempa}
```

```
2617   \endgroup
2618 }
2619 \let\mathdisplay@pop\@empty
2620 \def\mathdisplay@@pop{\the\mathdisplay@stack}
2621 \renewenvironment{equation}{%
2622   \incr@eqnum
2623   \mathdisplay@push
2624   \st@rredfalse \global\@eqnswtrue
2625   \mathdisplay{equation}%
2626 }{%
2627   \endmathdisplay{equation}%
2628   \mathdisplay@pop
2629   \ignorespacesafterend
2630 }
2631 \newenvironment{equation*}{%
2632   \mathdisplay@push
2633   \st@rredtrue \global\@eqnswfalse
2634   \mathdisplay{equation*}%
2635 }{%
2636   \endmathdisplay{equation*}%
2637   \mathdisplay@pop
2638   \ignorespacesafterend
2639 }
```

Note: LaTeX defines the `displaymath` environment in terms of `\[` and `\]`.

```
2640 \DeclareRobustCommand{\[}{\begin{equation*}}
2641 \DeclareRobustCommand{\]}{\end{equation*}}
```

The usual `\endinput` to ensure that random garbage at the end of the file doesn't get copied by `docstrip`.

```
2642 \endinput
```

17 Credits

Much of the code for the `amsmath` package had its orgin in `amstex.tex`, written by Michael Spivak. The initial work of porting `amstex.tex` to `amstex.sty` was done in 1988–1989 by Frank Mittelbach and Rainer Schöpf. In 1994 David M. Jones added the support for the `fleqn` option and did extensive improvements to the `align[at]` family of environments and to the equation number handling in general. Michael Downes at the AMS served as coordinator for the efforts of Mittelbach, Schöpf, and Jones, and has contributed various bug fixes and additional refinements over time.

Versions 1.0 and 1.1 of the package carried the name `amstex` instead of `amsmath`, to indicate its origins; the name was changed in 1994 to make it user-oriented rather than history-oriented.

Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.

A

B

T

U

V

W

X

Z