Bit pattern diagrams

Jean-Marc Bourguet
jm@bourguet.org

December 11, 2015

Abstract

The bitpattern package is designed to typeset bit patterns as they may appear in description of data format, hardware registers or transmission protocols. It covers thus more or less the same application domain as the package register and is somewhat related to bytefield.

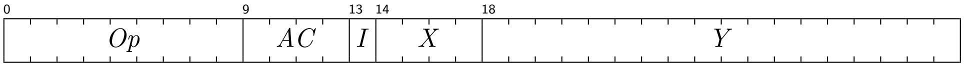
Comparared to register the formating is more compact, the syntax less verbose and bitpattern allows to choose between big endian and little endian bit numbering. But bitpattern is less well adapted to the use of long names for the fields and has no provision for a reset value. bytefield is more adequate to describe multi-word protocols, while bitpattern is more adapted to describe the intra-word structure of a single word.

1 Examples

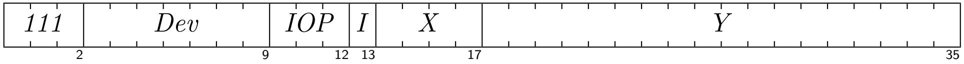
We'll the instruction formats of two processors to describe the features of bitpattern, the DEC PDP-10 and the Intel 8080.

The PDP-10 was a word-adressable 36-bit computer, something quite strange to our eyes (it's most familiar feature is probably the fact that it uses 2's complement) and the reason for which I've chosen it is that it was the computer on which TEX was first implemented.

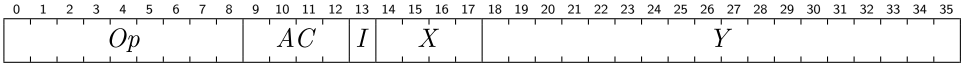
The PDP-10 has two instruction formats, one used for most of the instructions:



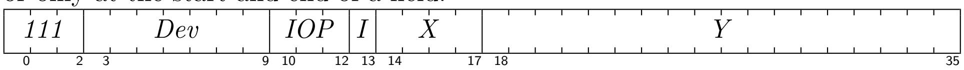
and the other used for IO instructions:



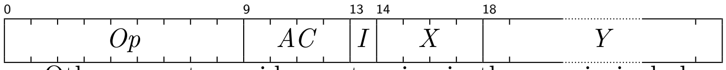
As well at putting the bit numbers above or below the field description, bitpattern allows to number all the bits:



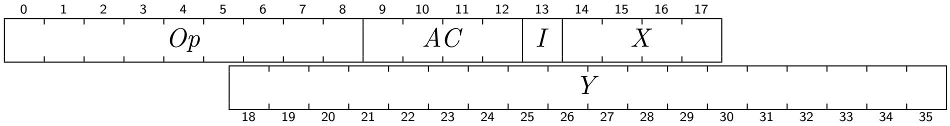
or only at the start and end of a field:



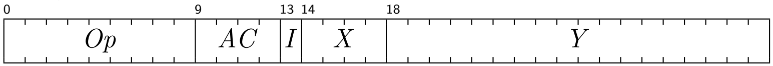
Having a long field like Y here may take more place than wanted. bitpattern allows to reduce that:



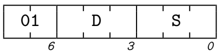
Other ways to avoid overstepping in the margin include splitting the pattern in two:

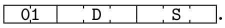


and reducing the width of the cells:



The PDP-10 numbered its bits in the big endian way. That's not the case of the Intel 8080. The 8080 was an 8-bit computer. Trying to pack instructions in 8-bit forces to use a lot of formats. One of them was used for the move instruction, and let's take that opportunity to start and compact the layout of the bit field by changing the format of the bit numbers and the field descriptions:



After that it is possible to still be more compact by removing the numbers, reducing the height of the ticks and avoiding the provide the space for ascender and descenders in the field name (obviously that is possible only with convenient field names). When that's done, you can put a bit pattern in a paragraph without disturbing it too much, like this . With some text afterwards to see how the next line is handled. The result seems quite readable, remark for instance that the baseline of the field description is aligned with the baseline of the line in which it is included.

2 Interface

2.1 `\bitpattern`

`\bitpattern` The `\bitpattern` macro is the macro which formats the patterns.

`\bitpattern[`⟨*format*⟩`]{`⟨*name*⟩`}[`⟨*size*⟩`][`⟨*width*⟩`]`. . . `/`

The first optional argument allows to control how the formatting is made and is the combination of the following keys:

`littleEndian` indicates that bit numbering is little endian, that is the leftward bit has the biggest number;

`bigEndian` indicates that bit numbering is big endian, that is the leftward bit has the lowest number;

`numberBitsAbove` indicates that the bit numbers should be put above the fields;

`numberBitsBelow` indicates that the bit numbers should be put below the fields;

`noBitNumbers` indicates that there should be no bit numbers;

`numberFieldsOnce` indicates that the fields should have only one bit number;

`numberFieldsTwice` indicates that the fields should have two bit numbers;

`numberAllBits` indicates that the fields shouldn't have a bit number indication;

`startBit=X` indicates that the number for the leftmost bit should be X;

`bitWidth=dimen` indicates that the size taken by a bit should be *dimen*;

`tickHeight=dimen` indicates that the size taken by the small ticks marking the bits should be *dimen.*

After the optional argument comes the description of the fields. Each field is described by its name as a mandatory argument (which does not have to be included in braces if it takes one character) followed by two optional arguments giving the size of the field (1 if omitted) and the width it should take (the same as its size if it is not specified).

Field descriptions are ended by a `/`.

2.2 Package options

The package `bitpattern` accept the following options which set up the default values for the formatting controls:

`littleEndian` indicates that the default for bit numbering is little endian, that is the leftward bit has the biggest number;

`bigEndian` indicates that the default for bit numbering is big endian, that is the leftward bit has the lowest number;

`numberBitsAbove` indicates that by default the bit numbers should be put above the fields;

`numberBitsBelow` indicates that by default the bit numbers should be put below the fields;

`noBitNumbers` indicates that by default there should be no bit numbers;

`numberFieldsOnce` indicates that by default the fields should have only one bit number;

`numberFieldsTwice` indicates that by default the fields should have two bit numbers;

`numberAllBits` indicates that by default the fields shouldn't have a bit number indication.

2.3 Commands controlling the format

\bpLittleEndian

\bpLittleEndian changes the default bit numbering to little endian.

\bpBigEndian

\bpBigEndian changes the default bit numbering to little endian.

\bpNumberBitsAbove

\bpNumberBitsAbove changes the default to having the numbering above the fields.

\bpNumberBitsBelow

\bpNumberBitsBelow changes the default to having the numbering below the fields.

\bpNoBitNumbers

\bpNoBitNumbers changes the default to having no numbering.

\bpNumberFieldsOnce

\bpNumberFieldsOnce changes the default to having the numbering done once per field.

\bpNumberFieldsTwice

\bpNumberFieldsTwice changes the default to having the numbering done twice per field.

\bpNumberAllBits

\bpNumberAllBits changes the default to having the numbering done for all bits of a field.

```
\bpStartAtBit{⟨number⟩}
```

\bpStartAtBit

\bpStartAtBit gives the default bit number of the leftmost bit.

```
\bpSetBitWidth{⟨length⟩}
```

\bpSetBitWidth

\bpSetBitWidth gives the default bit width

```
\bpSetTickHeight{⟨length⟩}
```

\bpSetTickHeight

\bpSetTickHeight gives the default height for the ticks marking the bits in a multi-bit field.

\bpFormatField

\bpFormatField is a one argument macro used fo format the field. It can be replaced. Care should be taken to format all the fields with the same height, so putting a \strut in the replacement is probably in order.

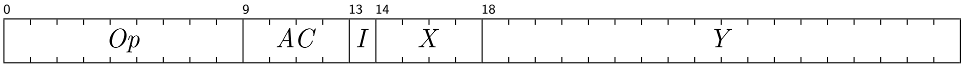
\bpFormatBitNumber

\bpFormatBitNumber is a one argument macro used fo format the bit numbers. It can be replaced. Care should be taken to format all the bit numbers with the same height, so putting a \strut in the replacement is perhaps in order.

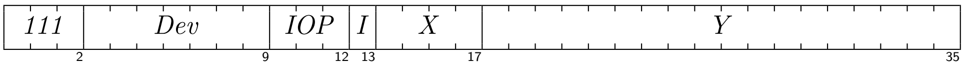
3 Examples revisited

bitpattern was loaded by:

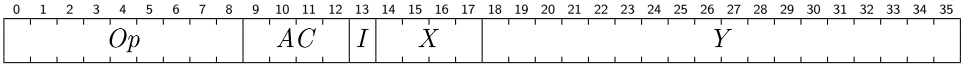
```
\RequirePackage[numberFieldsOnce,bigEndian,numberBitsAbove]{bitpattern}
```



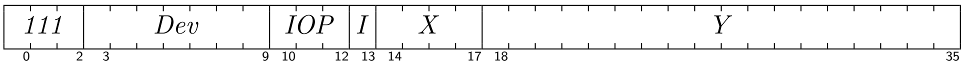
```
\bitpattern{Op}[9]{AC}[4]IX[4]Y[18]/
```



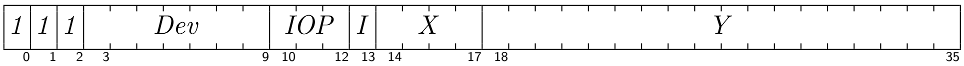
```
\bitpattern[numberBitsBelow]{111}[3]{Dev}[7]{IOP}[3]IX[4]Y[18]/
```



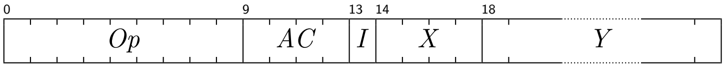
```
\bitpattern[numberAllBits]{Op}[9]{AC}[4]IX[4]Y[18]/
```



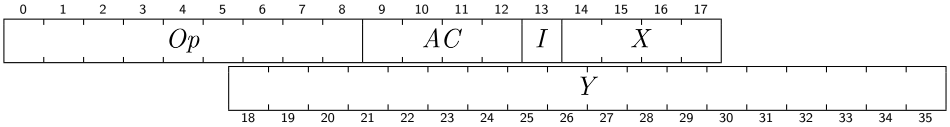
```
\bitpattern[numberBitsBelow,numberFieldsTwice]%
        {111}[3]{Dev}[7]{IOP}[3]IX[4]Y[18]/
```



```
\bitpattern[numberBitsBelow,numberFieldsTwice]%
        111{Dev}[7]{IOP}[3]IX[4]Y[18]/
```

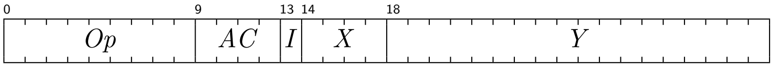


```
\bitpattern{Op}[9]{AC}[4]IX[4]Y[18][9]/
```

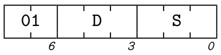


This wasn't composed directly by as two bit patterns, using the possibility of specifying a starting bit. To put a nicer touch, the width of the cells has been expanded (the default width had an overlap so small that going into the margin is a better format), and the numbering was put above the pattern for the first and below for the second.

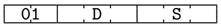
```
\noindent\bitpattern[bitWidth=1.5em,numberAllBits]{Op}[9]{AC}[4]IX[4]/ \\
\hspace*{1pt plus 1filll}
     \bitpattern[numberBitsBelow,startBit=18,bitWidth=1.5em,numberAllBits] Y[18]/
```



```
{\centering\bitpattern[bitWidth=0.8em]{Op}[9]{AC}[4]IX[4]Y[18]/ \par}
```



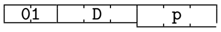
```
\renewcommand\bpFormatBitNumber[1]{{\tiny\ttfamily\emph{\strut#1}}}
\renewcommand\bpFormatField[1]{{\scriptsize\ttfamily\strut#1}}
{\centering\bitpattern[littleEndian,startBit=7,numberBitsBelow]{01}[2]{D}[3]{S}[3]/\par}
```



This example was made to be as compact as possible. Expanding the strut a little more vertically wouldn't be unwise.

```
\bpSetTickHeight{1pt}
\newlength{\bpDocExampleHeight}
\settoheight{\bpDocExampleHeight}{\scriptsize\ttfamily 0}
\newcommand{\bpDocExampleStrut}{\rule{0pt}{\bpDocExampleHeight}}
\renewcommand\bpFormatBitNumber[1]{{\tiny\ttfamily\emph{\strut#1}}}
\renewcommand\bpFormatField[1]{{\scriptsize\ttfamily\bpDocExampleStrut #1}}
{\centering\bitpattern[noBitNumbers,littleEndian,startBit=7]{01}[2]{D}[3]{S}[3]/\par}
```

Note that if you put a descender, the formatting is then disturbed:



It is to be noted that usually one does not change the formatting for every pattern, so setting the optional argument of `\bitpattern` is rarely used.