```
/* Copyright (c) 2002-2007  Marek Michalkiewicz
   Copyright (c) 2006, Carlos Lamas
   Copyright (c) 2009-2010, Jan Waclawek
   All rights reserved.

   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:

   * Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
   * Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in
     the documentation and/or other materials provided with the
     distribution.
   * Neither the name of the copyright holders nor the names of
     contributors may be used to endorse or promote products derived
     from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE. */

/* $Id$ */

/*
   pgmspace.h

   Contributors:
     Created by Marek Michalkiewicz <marekm@linux.org.pl>
     Eric B. Weddington <eric@ecentral.com>
     Wolfgang Haidinger <wh@vmars.tuwien.ac.at> (pgm_read_dword())
     Ivanov Anton <anton@arc.com.ru> (pgm_read_float())
 */

/** \file */
/** \defgroup avr_pgmspace <avr/pgmspace.h>: Program Space Utilities
    \code
    #include <avr/io.h>
    #include <avr/pgmspace.h>
    \endcode

    The functions in this module provide interfaces for a program to access
    data stored in program space (flash memory) of the device.  In order to
    use these functions, the target device must support either the \c LPM or
    \c ELPM instructions.

    \note These functions are an attempt to provide some compatibility with
    header files that come with IAR C, to make porting applications between
    different compilers easier.  This is not 100% compatibility though (GCC
    does not have full support for multiple address spaces yet).

    \note If you are working with strings which are completely based in ram,
    use the standard string functions described in \ref avr_string.

    \note If possible, put your constant tables in the lower 64 KB and use
    pgm_read_byte_near() or pgm_read_word_near() instead of
    pgm_read_byte_far() or pgm_read_word_far() since it is more efficient that
    way, and you can still use the upper 64K for executable code.
    All functions that are suffixed with a \c _P \e require their
    arguments to be in the lower 64 KB of the flash ROM, as they do
    not use ELPM instructions.  This is normally not a big concern as
    the linker setup arranges any program space constants declared
    using the macros from this header file so they are placed right after
    the interrupt vectors, and in front of any executable code.  However,
    it can become a problem if there are too many of these constants, or
    for bootloaders on devices with more than 64 KB of ROM.
    <em>All these functions will not work in that situation.</em>

    \note For <b>Xmega</b> devices, make sure the NVM controller
    command register (\c NVM.CMD or \c NVM_CMD) is set to 0x00 (NOP)
    before using any of these functions.
*/

#ifndef __PGMSPACE_H_
#define __PGMSPACE_H_ 1

#define __need_size_t
#include <inttypes.h>
#include <stddef.h>
#include <avr/io.h>

#ifndef __ATTR_CONST__
#define __ATTR_CONST__ __attribute__((__const__))
#endif

#ifndef __ATTR_PROGMEM__
#define __ATTR_PROGMEM__ __attribute__((__progmem__))
#endif

#ifndef __ATTR_PURE__
#define __ATTR_PURE__ __attribute__((__pure__))
#endif

/**
   \ingroup avr_pgmspace
   \def PROGMEM

   Attribute to use in order to declare an object being located in
   flash ROM.
 */
#define PROGMEM __ATTR_PROGMEM__

#ifdef __cplusplus
extern "C" {
#endif

#if defined(__DOXYGEN__)
/*
 * Doxygen doesn't grok the appended attribute syntax of
 * GCC, and confuses the typedefs with function decls, so
 * supply a doxygen-friendly view.
 */

/**
   \ingroup avr_pgmspace
   \typedef prog_void
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)
   Type of a "void" object located in flash ROM.  Does not make much
   sense by itself, but can be used to declare a "void *" object in
   flash ROM.
*/
typedef void PROGMEM prog_void;

/**
   \ingroup avr_pgmspace
   \typedef prog_char
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of a "char" object located in flash ROM.
*/
typedef char PROGMEM prog_char;

/**
   \ingroup avr_pgmspace
   \typedef prog_uchar
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "unsigned char" object located in flash ROM.
*/
typedef unsigned char PROGMEM prog_uchar;

/**
   \ingroup avr_pgmspace
   \typedef prog_int8_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "int8_t" object located in flash ROM.
*/
typedef int8_t PROGMEM prog_int8_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint8_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "uint8_t" object located in flash ROM.
*/
typedef uint8_t PROGMEM prog_uint8_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_int16_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "int16_t" object located in flash ROM.
*/
typedef int16_t PROGMEM prog_int16_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint16_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "uint16_t" object located in flash ROM.
*/
typedef uint16_t PROGMEM prog_uint16_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_int32_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "int32_t" object located in flash ROM.
*/
typedef int32_t PROGMEM prog_int32_t;

/**
```

```
   \ingroup avr_pgmspace
   \typedef prog_uint32_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "uint32_t" object located in flash ROM.
*/
typedef uint32_t PROGMEM prog_uint32_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_int64_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "int64_t" object located in flash ROM.

   \note This type is not available when the compiler
   option -mint8 is in effect.
*/
typedef int64_t PROGMEM prog_int64_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint64_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   #define directive, or by a -D compiler option.)

   Type of an "uint64_t" object located in flash ROM.

   \note This type is not available when the compiler
   option -mint8 is in effect.
*/
typedef uint64_t PROGMEM prog_uint64_t;

/** \ingroup avr_pgmspace
    \def PGM_P

    Used to declare a variable that is a pointer to a string in program
    space. */

#ifndef PGM_P
#define PGM_P const char *
```

```
#endif

/** \ingroup avr_pgmspace
    \def PGM_VOID_P

    Used to declare a generic pointer to an object in program space. */

#ifndef PGM_VOID_P
#define PGM_VOID_P const void *
#endif

#elif defined(__PROG_TYPES_COMPAT__)  /* !DOXYGEN */

typedef void prog_void __attribute__((__progmem__,deprecated("prog_void type is
deprecated.")));
typedef char prog_char __attribute__((__progmem__,deprecated("prog_char type is
deprecated.")));
typedef unsigned char prog_uchar __attribute__((__progmem__,deprecated("prog_uchar
type is deprecated.")));
typedef int8_t    prog_int8_t   __attribute__((__progmem__,deprecated("prog_int8_t
type is deprecated.")));
typedef uint8_t   prog_uint8_t  __attribute__((__progmem__,deprecated("prog_uint8_t
type is deprecated.")));
typedef int16_t   prog_int16_t  __attribute__((__progmem__,deprecated("prog_int16_t
type is deprecated.")));
typedef uint16_t  prog_uint16_t __attribute__((__progmem__,deprecated("prog_uint16_t
type is deprecated.")));
typedef int32_t   prog_int32_t  __attribute__((__progmem__,deprecated("prog_int32_t
type is deprecated.")));
typedef uint32_t  prog_uint32_t __attribute__((__progmem__,deprecated("prog_uint32_t
type is deprecated.")));
#if !__USING_MINT8
typedef int64_t   prog_int64_t  __attribute__((__progmem__,deprecated("prog_int64_t
type is deprecated.")));
typedef uint64_t  prog_uint64_t __attribute__((__progmem__,deprecated("prog_uint64_t
type is deprecated.")));
#endif

#ifndef PGM_P
#define PGM_P const prog_char *
#endif

#ifndef PGM_VOID_P
#define PGM_VOID_P const prog_void *
#endif

#else /* !defined(__DOXYGEN__), !defined(__PROG_TYPES_COMPAT__) */

#ifndef PGM_P
#define PGM_P const char *
#endif

#ifndef PGM_VOID_P
#define PGM_VOID_P const void *
#endif
#endif /* defined(__DOXYGEN__), defined(__PROG_TYPES_COMPAT__) */

/* Although in C, we can get away with just using __c, it does not work in
   C++. We need to use &__c[0] to avoid the compiler puking. Dave Hylands
   explaned it thusly,

     Let's suppose that we use PSTR("Test"). In this case, the type returned
     by __c is a prog_char[5] and not a prog_char *. While these are
     compatible, they aren't the same thing (especially in C++). The type
     returned by &__c[0] is a prog_char *, which explains why it works
     fine. */
```

```
#if defined(__DOXYGEN__)
/*
 * The #define below is just a dummy that serves documentation
 * purposes only.
 */
/** \ingroup avr_pgmspace
    \def PSTR(s)

    Used to declare a static pointer to a string in program space. */
# define PSTR(s) ((const PROGMEM char *)(s))
#else  /* !DOXYGEN */
/* The real thing. */
# define PSTR(s) (__extension__({static const char __c[] PROGMEM = (s); &__c[0];}))
#endif /* DOXYGEN */

#define __LPM_classic__(addr)   \
(__extension__({                \
    uint16_t __addr16 = (uint16_t)(addr); \
    uint8_t __result;           \
    __asm__ __volatile__        \
    (                           \
        "lpm" "\n\t"            \
        "mov %0, r0" "\n\t"     \
        : "=r" (__result)       \
        : "z" (__addr16)        \
        : "r0"                  \
    );                          \
    __result;                   \
}))

#define __LPM_tiny__(addr)      \
(__extension__({                \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    uint8_t __result;           \
    __asm__                     \
    (                           \
        "ld %0, z" "\n\t"       \
        : "=r" (__result)       \
        : "z" (__addr16)        \
    );                          \
    __result;                   \
}))

#define __LPM_enhanced__(addr)  \
(__extension__({                \
    uint16_t __addr16 = (uint16_t)(addr); \
    uint8_t __result;           \
    __asm__ __volatile__        \
    (                           \
        "lpm %0, Z" "\n\t"      \
        : "=r" (__result)       \
        : "z" (__addr16)        \
    );                          \
    __result;                   \
}))

#define __LPM_word_classic__(addr)          \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint16_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm"           "\n\t"              \
        "mov %A0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %B0, r0"   "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
        : "r0"                              \
    );                                      \
    __result;                               \
}))

#define __LPM_word_tiny__(addr)             \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    uint16_t __result;                      \
    __asm__                                 \
    (                                       \
        "ld %A0, z+"     "\n\t"             \
        "ld %B0, z"      "\n\t"             \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_word_enhanced__(addr)         \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint16_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm %A0, Z+"   "\n\t"              \
        "lpm %B0, Z"    "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_dword_classic__(addr)         \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint32_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm"           "\n\t"              \
        "mov %A0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %B0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %C0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %D0, r0"   "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
        : "r0"                              \
    );                                      \
    __result;                               \
}))

#define __LPM_dword_tiny__(addr)            \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    uint32_t __result;                      \
    __asm__                                 \
    (                                       \
        "ld %A0, z+"    "\n\t"              \
        "ld %B0, z+"    "\n\t"              \
        "ld %C0, z+"    "\n\t"              \
        "ld %D0, z"     "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_dword_enhanced__(addr)        \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint32_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm %A0, Z+"   "\n\t"              \
        "lpm %B0, Z+"   "\n\t"              \
        "lpm %C0, Z+"   "\n\t"              \
        "lpm %D0, Z"    "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_float_classic__(addr)         \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    float __result;                         \
    __asm__ __volatile__                    \
    (                                       \
        "lpm"           "\n\t"              \
        "mov %A0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %B0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %C0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %D0, r0"   "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
        : "r0"                              \
    );                                      \
    __result;                               \
}))

#define __LPM_float_tiny__(addr)            \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    float __result;                         \
    __asm__                                 \
    (                                       \
        "ld %A0, z+"   "\n\t"               \
        "ld %B0, z+"   "\n\t"               \
        "ld %C0, z+"   "\n\t"               \
        "ld %D0, z"    "\n\t"               \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_float_enhanced__(addr)        \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    float __result;                         \
    __asm__ __volatile__                    \
    (                                       \
        "lpm %A0, Z+"   "\n\t"              \
        "lpm %B0, Z+"   "\n\t"              \
        "lpm %C0, Z+"   "\n\t"              \
        "lpm %D0, Z"    "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#if defined (__AVR_HAVE_LPMX__)
#define __LPM(addr)         __LPM_enhanced__(addr)
#define __LPM_word(addr)    __LPM_word_enhanced__(addr)
#define __LPM_dword(addr)   __LPM_dword_enhanced__(addr)
#define __LPM_float(addr)   __LPM_float_enhanced__(addr)
/*
Macro to read data from program memory for avr tiny parts(tiny 4/5/9/10/20/40).
why:
- LPM instruction is not available in AVR_TINY instruction set.
- Programs are executed starting from address 0x0000 in program memory.
But it must be addressed starting from 0x4000 when accessed via data memory.
Reference: TINY device (ATTiny 4,5,9,10,20 and 40) datasheets
Bug: avrtc-536
*/
#elif defined (__AVR_TINY__)
#define __LPM(addr)         __LPM_tiny__(addr)
#define __LPM_word(addr)    __LPM_word_tiny__(addr)
#define __LPM_dword(addr)   __LPM_dword_tiny__(addr)
#define __LPM_float(addr)   __LPM_float_tiny__(addr)
#else
#define __LPM(addr)         __LPM_classic__(addr)
#define __LPM_word(addr)    __LPM_word_classic__(addr)
#define __LPM_dword(addr)   __LPM_dword_classic__(addr)
#define __LPM_float(addr)   __LPM_float_classic__(addr)
#endif

/** \ingroup avr_pgmspace
    \def pgm_read_byte_near(address_short)
    Read a byte from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_byte_near(address_short) __LPM((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_word_near(address_short)
    Read a word from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_word_near(address_short) __LPM_word((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_dword_near(address_short)
    Read a double word from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_dword_near(address_short) \
    __LPM_dword((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_float_near(address_short)
    Read a float from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_float_near(address_short) \
    __LPM_float((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_ptr_near(address_short)
    Read a pointer from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_ptr_near(address_short) \
    (void*)__LPM_word((uint16_t)(address_short))

#if defined(RAMPZ) || defined(__DOXYGEN__)

/* Only for devices with more than 64K of program memory.
   RAMPZ must be defined (see iom103.h, iom128.h).
*/

/* The classic functions are needed for ATmega103. */

#define __ELPM_classic__(addr)      \
(__extension__({                    \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;               \
    __asm__ __volatile__            \
    (                               \
        "out %2, %C1" "\n\t"        \
        "mov r31, %B1" "\n\t"       \
        "mov r30, %A1" "\n\t"       \
        "elpm" "\n\t"               \
        "mov %0, r0" "\n\t"         \
        : "=r" (__result)           \
        : "r" (__addr32),           \
          "I" (_SFR_IO_ADDR(RAMPZ)) \
        : "r0", "r30", "r31"        \
    );                              \
    __result;                       \
}))

#define __ELPM_enhanced__(addr)     \
(__extension__({                    \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;               \
    __asm__ __volatile__            \
    (                               \
        "out %2, %C1" "\n\t"        \
        "movw r30, %1" "\n\t"       \
        "elpm %0, Z+" "\n\t"        \
        : "=r" (__result)           \
        : "r" (__addr32),           \
          "I" (_SFR_IO_ADDR(RAMPZ)) \
        : "r30", "r31"              \
    );                              \
    __result;                       \
}))

#define __ELPM_xmega__(addr)        \
(__extension__({                    \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;               \
    __asm__ __volatile__            \
    (                               \
        "in __tmp_reg__, %2" "\n\t" \
        "out %2, %C1" "\n\t"        \
        "movw r30, %1" "\n\t"       \
        "elpm %0, Z+" "\n\t"        \
        "out %2, __tmp_reg__"       \
        : "=r" (__result)           \
        : "r" (__addr32),           \
          "I" (_SFR_IO_ADDR(RAMPZ)) \
        : "r30", "r31"              \
    );                              \
    __result;                       \
}))

#define __ELPM_word_classic__(addr)     \
(__extension__({                        \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint16_t __result;                  \
    __asm__ __volatile__                \
    (                                   \
        "out %2, %C1"   "\n\t"          \
        "mov r31, %B1"  "\n\t"          \
        "mov r30, %A1"  "\n\t"          \
        "elpm"          "\n\t"          \
        "mov %A0, r0"   "\n\t"          \
        "in r0, %2"     "\n\t"          \
        "adiw r30, 1"   "\n\t"          \
        "adc r0, __zero_reg__" "\n\t"   \
        "out %2, r0"    "\n\t"          \
        "elpm"          "\n\t"          \
        "mov %B0, r0"   "\n\t"          \
        : "=r" (__result)               \
        : "r" (__addr32),               \
          "I" (_SFR_IO_ADDR(RAMPZ))     \
        : "r0", "r30", "r31"            \
    );                                  \
    __result;                           \
}))

#define __ELPM_word_enhanced__(addr)    \
(__extension__({                        \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint16_t __result;                  \
    __asm__ __volatile__                \
    (                                   \
        "out %2, %C1"   "\n\t"          \
        "movw r30, %1"  "\n\t"          \
        "elpm %A0, Z+"  "\n\t"          \
        "elpm %B0, Z"   "\n\t"          \
        : "=r" (__result)               \
        : "r" (__addr32),               \
          "I" (_SFR_IO_ADDR(RAMPZ))     \
        : "r30", "r31"                  \
    );                                  \
    __result;                           \
}))

#define __ELPM_word_xmega__(addr)       \
(__extension__({                        \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint16_t __result;                  \
    __asm__ __volatile__                \
    (                                   \
        "in __tmp_reg__, %2" "\n\t"     \
        "out %2, %C1"   "\n\t"          \
        "movw r30, %1"  "\n\t"          \
        "elpm %A0, Z+"  "\n\t"          \
        "elpm %B0, Z"   "\n\t"          \
        "out %2, __tmp_reg__"           \
        : "=r" (__result)               \
        : "r" (__addr32),               \
          "I" (_SFR_IO_ADDR(RAMPZ))     \
        : "r30", "r31"                  \
    );                                  \
    __result;                           \
}))

#define __ELPM_dword_classic__(addr)      \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint32_t __result;                    \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"          "\n\t"     \
        "mov r31, %B1"         "\n\t"     \
        "mov r30, %A1"         "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %A0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %B0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %C0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %D0, r0"          "\n\t"     \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r0", "r30", "r31"              \
    );                                    \
    __result;                             \
}))

#define __ELPM_dword_enhanced__(addr)     \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint32_t __result;                    \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

#define __ELPM_dword_xmega__(addr)        \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint32_t __result;                     \
    __asm__ __volatile__                   \
    (                                      \
        "in __tmp_reg__, %2" "\n\t"        \
        "out %2, %C1"   "\n\t"             \
        "movw r30, %1"  "\n\t"             \
        "elpm %A0, Z+"  "\n\t"             \
        "elpm %B0, Z+"  "\n\t"             \
        "elpm %C0, Z+"  "\n\t"             \
        "elpm %D0, Z"   "\n\t"             \
        "out %2, __tmp_reg__"              \
        : "=r" (__result)                  \
        : "r" (__addr32),                  \
          "I" (_SFR_IO_ADDR(RAMPZ))        \
        : "r30", "r31"                     \
    );                                     \
    __result;                              \
}))

#define __ELPM_float_classic__(addr)      \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    float __result;                       \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"          "\n\t"     \
        "mov r31, %B1"         "\n\t"     \
        "mov r30, %A1"         "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %A0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %B0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %C0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %D0, r0"          "\n\t"     \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r0", "r30", "r31"              \
    );                                    \
    __result;                             \
}))

#define __ELPM_float_enhanced__(addr)     \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    float __result;                       \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

#define __ELPM_float_xmega__(addr)        \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    float __result;                       \
    __asm__ __volatile__                  \
    (                                     \
        "in __tmp_reg__, %2" "\n\t"       \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        "out %2, __tmp_reg__"             \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

/*
Check for architectures that implement RAMPD (avrxmega3, avrxmega5,
avrxmega7) as they need to save/restore RAMPZ for ELPM macros so it does
not interfere with data accesses.
*/
#if defined (__AVR_HAVE_RAMPD__)

#define __ELPM(addr)        __ELPM_xmega__(addr)
#define __ELPM_word(addr)   __ELPM_word_xmega__(addr)
#define __ELPM_dword(addr)  __ELPM_dword_xmega__(addr)
#define __ELPM_float(addr)  __ELPM_float_xmega__(addr)

#else

#if defined (__AVR_HAVE_LPMX__)

#define __ELPM(addr)        __ELPM_enhanced__(addr)
#define __ELPM_word(addr)   __ELPM_word_enhanced__(addr)
#define __ELPM_dword(addr)  __ELPM_dword_enhanced__(addr)
#define __ELPM_float(addr)  __ELPM_float_enhanced__(addr)

#else

#define __ELPM(addr)        __ELPM_classic__(addr)
#define __ELPM_word(addr)   __ELPM_word_classic__(addr)
#define __ELPM_dword(addr)  __ELPM_dword_classic__(addr)
#define __ELPM_float(addr)  __ELPM_float_classic__(addr)

#endif  /* __AVR_HAVE_LPMX__ */

#endif  /* __AVR_HAVE_RAMPD__ */


/** \ingroup avr_pgmspace
    \def pgm_read_byte_far(address_long)
    Read a byte from the program space with a 32-bit (far) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_byte_far(address_long)  __ELPM((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_word_far(address_long)
    Read a word from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_word_far(address_long)  __ELPM_word((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_dword_far(address_long)
    Read a double word from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_dword_far(address_long) __ELPM_dword((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_float_far(address_long)
    Read a float from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_float_far(address_long) __ELPM_float((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_ptr_far(address_long)
    Read a pointer from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_ptr_far(address_long) (void*)__ELPM_word((uint32_t)(address_long))

#endif /* RAMPZ or __DOXYGEN__ */

/** \ingroup avr_pgmspace
    \def pgm_read_byte(address_short)
    Read a byte from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_byte(address_short)    pgm_read_byte_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_word(address_short)
    Read a word from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_word(address_short)    pgm_read_word_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_dword(address_short)
    Read a double word from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_dword(address_short)   pgm_read_dword_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_float(address_short)
    Read a float from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_float(address_short)   pgm_read_float_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_ptr(address_short)
    Read a pointer from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_ptr(address_short)     pgm_read_ptr_near(address_short)

/* pgm_get_far_address() macro

   This macro facilitates the obtention of a 32 bit "far" pointer (only 24 bits
   used) to data even passed the 64KB limit for the 16 bit ordinary pointer. It
   is similar to the '&' operator, with some limitations.

   Comments:

   - The overhead is minimal and it's mainly due to the 32 bit size operation.

   - 24 bit sizes guarantees the code compatibility for use in future devices.

   - hh8() is an undocumented feature but seems to give the third significant byte
     of a 32 bit data and accepts symbols, complementing the functionality of hi8()
     and lo8(). There is not an equivalent assembler function to get the high
     significant byte.

   - 'var' has to be resolved at linking time as an existing symbol, i.e, a simple
     type variable name, an array name (not an indexed element of the array, if the
     index is a constant the compiler does not complain but fails to get the address
     if optimization is enabled), a struct name or a struct field name, a function
     identifier, a linker defined identifier,...

   - The returned value is the identifier's VMA (virtual memory address) determined
     by the linker and falls in the corresponding memory region. The AVR Harvard
     architecture requires non overlapping VMA areas for the multiple address spaces
     in the processor: Flash ROM, RAM, and EEPROM. Typical offset for this are
     0x00000000, 0x00800xx0, and 0x00810000 respectively, derived from the linker
   script used and linker options. The value returned can be seen then as a
     universal pointer.

*/

#define pgm_get_far_address(var)                          \
({                                                    \
  uint_farptr_t tmp;                                \
                                                    \
  __asm__ __volatile__(                             \
                                                    \
      "ldi  %A0, lo8(%1)"           "\n\t"    \
      "ldi  %B0, hi8(%1)"           "\n\t"    \
      "ldi  %C0, hh8(%1)"           "\n\t"    \
      "clr  %D0"                    "\n\t"    \
    :                                             \
      "=d" (tmp)                                \
    :                                              \
      "p"  (&(var))                              \
  );                                                 \
  tmp;                                               \
})



extern const void * memchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;
extern int memcmp_P(const void *, const void *, size_t) __ATTR_PURE__;
extern void *memccpy_P(void *, const void *, int __val, size_t);
extern void *memcpy_P(void *, const void *, size_t);
extern void *memmem_P(const void *, size_t, const void *, size_t) __ATTR_PURE__;
extern const void * memrchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;
extern char *strcat_P(char *, const char *);
extern const char * strchr_P(const char *, int __val) __ATTR_CONST__;
extern const char * strchrnul_P(const char *, int __val) __ATTR_CONST__;
extern int strcmp_P(const char *, const char *) __ATTR_PURE__;
extern char *strcpy_P(char *, const char *);
extern int strcasecmp_P(const char *, const char *) __ATTR_PURE__;
extern char *strcasestr_P(const char *, const char *) __ATTR_PURE__;
extern size_t strcspn_P(const char *__s, const char * __reject) __ATTR_PURE__;
extern size_t strlcat_P (char *, const char *, size_t );
extern size_t strlcpy_P (char *, const char *, size_t );
extern size_t __strlen_P(const char *) __ATTR_CONST__;  /* program memory can't change
*/
extern size_t strnlen_P(const char *, size_t) __ATTR_CONST__; /* program memory can't
change */
extern int strncmp_P(const char *, const char *, size_t) __ATTR_PURE__;
extern int strncasecmp_P(const char *, const char *, size_t) __ATTR_PURE__;
extern char *strncat_P(char *, const char *, size_t);
extern char *strncpy_P(char *, const char *, size_t);
extern char *strpbrk_P(const char *__s, const char * __accept) __ATTR_PURE__;
extern const char * strrchr_P(const char *, int __val) __ATTR_CONST__;
extern char *strsep_P(char **__sp, const char * __delim);
extern size_t strspn_P(const char *__s, const char * __accept) __ATTR_PURE__;
extern char *strstr_P(const char *, const char *) __ATTR_PURE__;
extern char *strtok_P(char *__s, const char * __delim);
extern char *strtok_rP(char *__s, const char * __delim, char **__last);

extern size_t strlen_PF (uint_farptr_t src) __ATTR_CONST__; /* program memory can't
change */
extern size_t strnlen_PF (uint_farptr_t src, size_t len) __ATTR_CONST__; /* program
memory can't change */
extern void *memcpy_PF (void *dest, uint_farptr_t src, size_t len);
extern char *strcpy_PF (char *dest, uint_farptr_t src);
extern char *strncpy_PF (char *dest, uint_farptr_t src, size_t len);
extern char *strcat_PF (char *dest, uint_farptr_t src);
extern size_t strlcat_PF (char *dst, uint_farptr_t src, size_t siz);
extern char *strncat_PF (char *dest, uint_farptr_t src, size_t len);
extern int strcmp_PF (const char *s1, uint_farptr_t s2) __ATTR_PURE__;
extern int strncmp_PF (const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;
extern int strcasecmp_PF (const char *s1, uint_farptr_t s2) __ATTR_PURE__;
extern int strncasecmp_PF (const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;
extern char *strstr_PF (const char *s1, uint_farptr_t s2);
extern size_t strlcpy_PF (char *dst, uint_farptr_t src, size_t siz);
extern int memcmp_PF(const void *, uint_farptr_t, size_t) __ATTR_PURE__;


__attribute__((__always_inline__)) static inline size_t strlen_P(const char * s);
static inline size_t strlen_P(const char *s) {
  return __builtin_constant_p(__builtin_strlen(s))
     ? __builtin_strlen(s) : __strlen_P(s);
}
```

```
#ifdef __cplusplus
}
#endif

#endif /* __PGMSPACE_H_ */
```