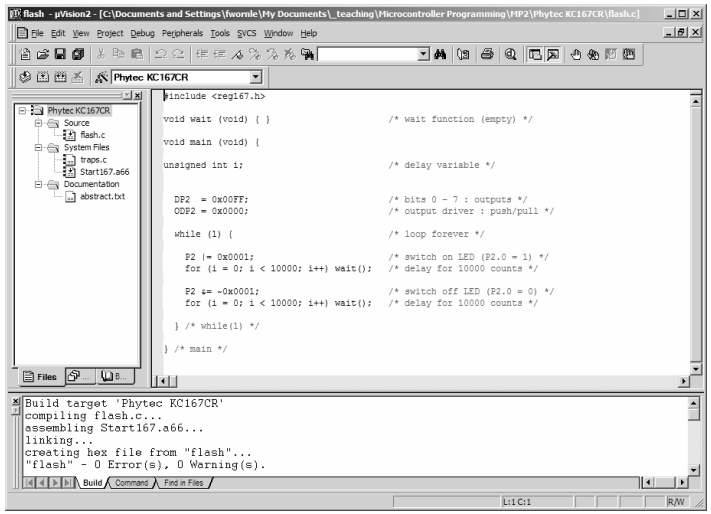
week	lecture	topics
3	Compilation process – fundamentals	- Compiler, assembler, linker - The build process in detail - Compiler options - Linker options - Warnings and error messages - Example: Keil C166 projects - Source level debugging using a micro-controller simulator - Download into target RAM - Source level debugging using a target Monitor

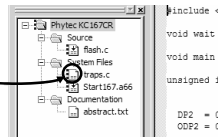
Compilation process example: flashing LED



Compilation process example: flashing LED

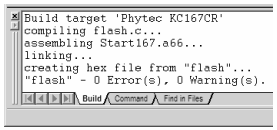
Source files:

- flash.c … the program
- traps.c … interrupts (unused)
- start167.a66 … µC initializations



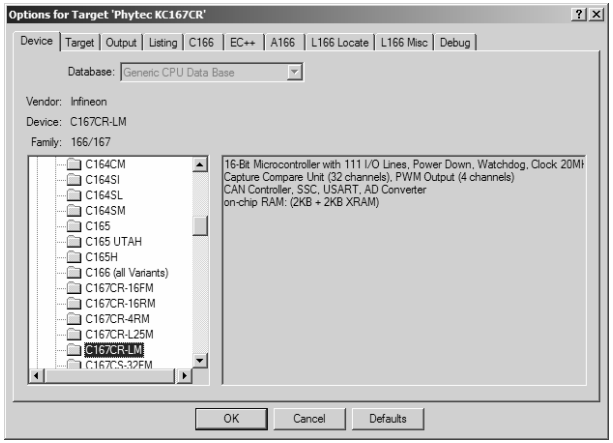
Build process:

- compiling flash.c
- assembling Start167.a66
- linking both objects to ‘flash’ and producing EEPROM ‘hex’ file



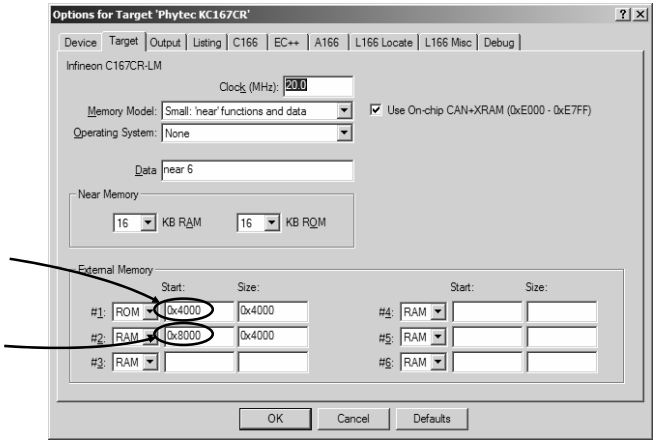
Compilation process example: flashing LED

Device options: Select micro-controller core (here: C167CR-LM) for this project



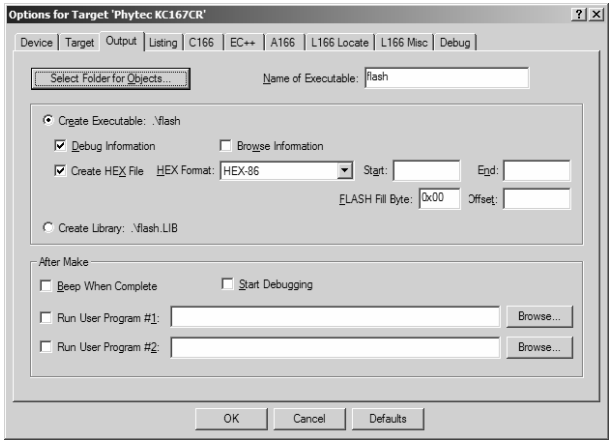
Compilation process example: flashing LED

Target options: CPU, external memory (CODE at 0x4000, DATA at 0x8000), etc.



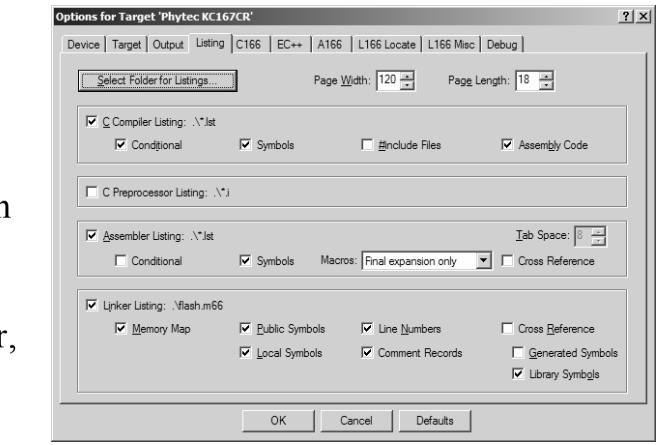
Compilation process example: flashing LED

Output options: Configure executable output file (*flash*) and the down-loadable 'hex' file (*flash.h86*)



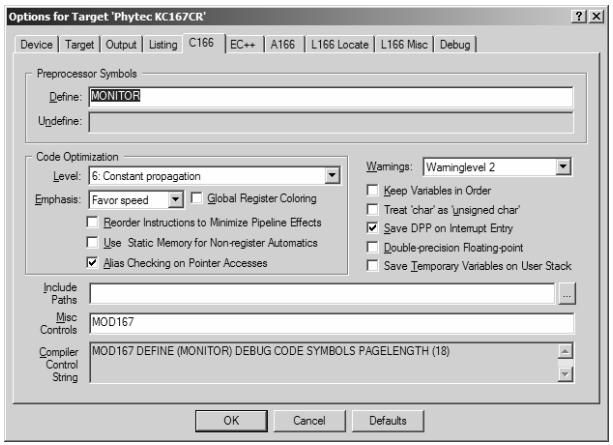
Compilation process example: flashing LED

Listing options: Specify detail of information within the listing files of compiler, assembler, linker, …



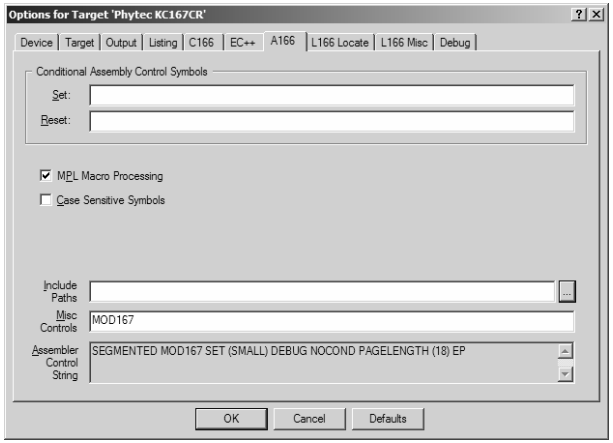
Compilation process example: flashing LED

Compiler options: Define macros (e.g. MONITOR), level of feedback (warnings, debug info), etc.



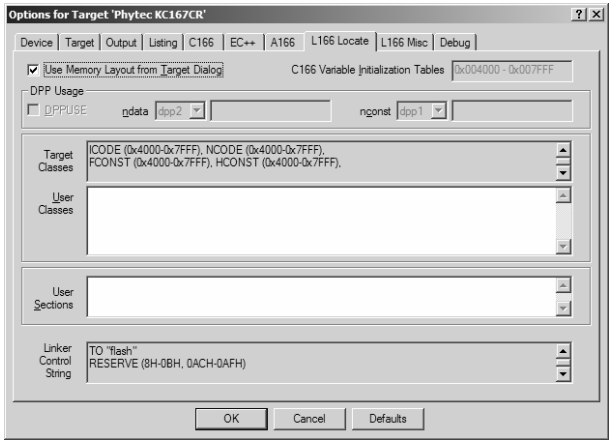
Compilation process example: flashing LED

Assembler options: Memory model (here: SMALL – near code, near data), CPU details (MOD167), etc.



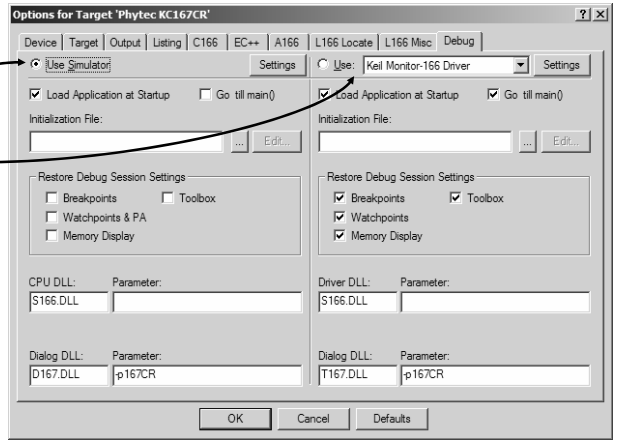
Compilation process example: flashing LED

Linker and Locator options: Specify memory map of the executable (RAM, ROM, etc.)



Compilation process example: flashing LED

Debugging: *Simulator* (PC based) or *Monitor* (download to target board, debugging via serial interface



Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                        /* wait function (empty) */

void main (void) {

unsigned int i;                             /* delay variable */


  DP2  = 0x00FF;                            /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                            /* output driver : push/pull */

  while (1) {                               /* loop forever */

    P2 |= 0x0001;                           /* switch on LED (P2.0 = 1) */
    for (i = 0; i < 10000; i++) wait();     /* delay for 10000 counts */

    P2 &= ~0x0001;                          /* switch off LED (P2.0 = 0) */
    for (i = 0; i < 10000; i++) wait();     /* delay for 10000 counts */

  } /* while(1) */

} /* main */
```

Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                    /* wait function (empty) */

void main (void) {

unsigned int i;                         /* delay variable */


  DP2  = 0x00FF;                        /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                        /* output driver : push/pull */


    (...)

}
```

Macro definitions of *special function registers* (SFR) such as DP2 (data direction, port 2) or ODP2 (output driver, port 2)

Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                    /* wait function (empty) */

void main (void) {

unsigned int i;                         /* delay variable */


  DP2  = 0x00FF;                        /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                        /* output driver : push/pull */


    (...)

}
```

Empty function 'wait'; calling upon wait doesn't do anything but waste a little bit of CPU time. This is an easy way of slowing down the processor

Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                    /* wait function (empty) */

void main (void) {

unsigned int i;                         /* delay variable */


  DP2  = 0x00FF;                        /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                        /* output driver : push/pull */


    (...)

}
```

Main program with no call-up parameters [*(void)* ... on the right-hand side of *main*] and no return values [*void* … on the left-hand side of *main*].

Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                    /* wait function (empty) */

void main (void) {

unsigned int i;                         /* delay variable */


  DP2  = 0x00FF;                        /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                        /* output driver : push/pull */


    (...)

}
```

Variable, local to *main*. This is a 16-bit unsigned integer variable which can store data values ranging from 0 to 2^{16}-1 = 65535 = 0xFFFF (hex)

Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                    /* wait function (empty) */

void main (void) {

unsigned int i;                         /* delay variable */


  DP2  = 0x00FF;                        /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                        /* output driver : push/pull */


    (...)

}
```

Assignment of values 0x00FF (= binary pattern 0000.0000.1111.1111) and 0x0000 (= 0) to special function registers DP2 and ODP2, respectively

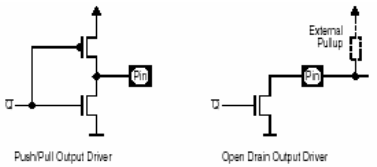
Compilation process example: flashing LED

```
#include <reg167.h>

void wait (void) { }                    /* wait function (empty) */

void main (void) {

unsigned int i;                         /* delay variable */


  DP2  = 0x00FF;                        /* bits 0 - 7 : outputs */
  ODP2 = 0x0000;                        /* output driver : push/pull */


    (...)

}
```

This programs bits 0 – 7 of port 2 as outputs (DP2 = 0000.0000.1111.1111) and selects the push/pull output driver of port 2 (ODP2 = 0x0000)

Compilation process example: flashing LED

- Each digital output can be configured to be driven by a push/pull amplifier or an open-drain transistor



- Push/pull amplifiers can drive the output to logical high or logical low; open-drain outputs can only pull the output to ground (GND) or leave it in a *floating* state (high impedance, *Hi-Z*)

Compilation process example: flashing LED

- Implemented using bipolar transistors, push/pull technology is also referred to as *Transistor-Transistor Logic (TTL)*; the equivalent field effect transistor (FET) circuitry is called *Complementary Metal-Oxide-Silicon logic (CMOS)*
- Open-drain (open-collector) technology requires an external *pull-up resistor* to allow the output to attain a logical high level; open-drain circuits are useful when multiple outputs are to be tied together in parallel to implement a wired-AND function

Compilation process example: flashing LED

```
(...)

while (1) {                          /* loop forever */

  P2 |= 0x0001;                      /* switch on LED (P2.0 = 1) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

  P2 &= ~0x0001;                     /* switch off LED (P2.0 = 0) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

} /* while(1) */

} /* main */
```

Embedded programs never end; this is usually achieved with an endless loop, e.g. *while(1){...}* or *for(;;){ ... }* – both constructs are *unconditional*

Compilation process example: flashing LED

```
(...)

while (1) {                          /* loop forever */

  P2 |= 0x0001;                      /* switch on LED (P2.0 = 1) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

  P2 &= ~0x0001;                     /* switch off LED (P2.0 = 0) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

} /* while(1) */

} /* main */
```

P2 |= 0x0001 is equivalent to *P2 = P2 | 0x0001*. This combines the contents of port 2 with the mask 0x0001 (logical OR); effectively this lights the LED connected to bit 0 of port 2 (P2.0 = 1)

Compilation process example: flashing LED

```
(...)

while (1) {                          /* loop forever */

  P2 |= 0x0001;                      /* switch on LED (P2.0 = 1) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

  P2 &= ~0x0001;                     /* switch off LED (P2.0 = 0) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

} /* while(1) */

} /* main */
```

Dummy function call to empty function *wait*. This wastes some CPU time. Doing this inside a loop is a simple way of producing a short time delay

Compilation process example: flashing LED

```
(...)

while (1) {                          /* loop forever */

  P2 |= 0x0001;                      /* switch on LED (P2.0 = 1) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

  P2 &= ~0x0001;                     /* switch off LED (P2.0 = 0) */
  for (i = 0; i < 10000; i++) wait();   /* delay for 10000 counts */

} /* while(1) */

} /* main */
```

*P2 **&**= ~0x0001* is short for *P2 = P2 **&** 0xFFFE*. The mask *~0x0001 (**NOT** 0000.0000.0000.0001)* expands to *1111.1111.1111.1110 = 0xFFFE*. The ***AND*** operation thus clears bit 0 of port 2 (LED off)

Compilation process example: flashing LED

```
      (…)

  while (1) {                           /* loop forever */

    P2 |= 0x0001;                       /* switch on LED (P2.0 = 1) */
    for (i = 0; i < 10000; i++) wait();  /* delay for 10000 counts */

    P2 &= ~0x0001;                      /* switch off LED (P2.0 = 0) */
    for (i = 0; i < 10000; i++) wait();  /* delay for 10000 counts */

  } /* while(1) */

} /* main */
```

More time wasting to make the *off-phase* of the LED as long as the *on-phase*. Altogether, the program produces a slowly flashing LED

Compilation process example: flashing LED

- Compilation of this program using the KEIL tool chain (µVision) produces the following files:
  - flash.lst assembler listing of *flash.c*
  - flash.obj object file (machine code)
  - Start167.lst assembler listing of *Start167.a66*
  - Start167.obj object file (machine code)
  - flash.lnp linker command file
  - flash.m66 detailed memory map
  - flash linked executable module
  - flash.h86 EEPROM version of *flash*

Compilation process example: flashing LED

- Other compilers (e.g. GNU gcc) might produce a slightly different set of output files
- Understanding the purpose and contents of these files is imperative to the successful development of embedded microcontroller programs
- The tools used to produce these files can be called upon from within the KEIL integrated development environment (IDE) or from a shell (e.g. MSDOS command line prompt, MATLAB command line prompt, UNIX/Linux terminal, etc.)

Compilation process example: flashing LED

- On PCs the KEIL tool chain includes:
  - C166.exe ANSI-C cross-compiler (C166)
  - EC166.exe embedded C++ cross-compiler
  - A166.exe macro assembler (C166)
  - L166.exe linker and locator
  - LIB166.exe library manager utility
  - OH166.exe object to hex-file converter
- Not all of these programs are used every time a program is compiled (built)

Compilation process example: flashing LED

- The flashing LED example makes use of…
  - C166 … to compile the C-source code into assembler code for the C167 µC
  - A166 … to turn the assembler code into relocatable machine code (object file)
  - L166 … to link all object files (flash.obj, Start167.obj) to an absolute executable
  - OH166 … to produce an INTEL hex-86 file which can be written to an EEPROM using a FLASH/EEPROM burner

Compilation process example: flashing LED

- Listing file *flash.lst*:

```
C166 COMPILER V5.03, FLASH
10/16/2004 17:52:41 PAGE 1


C166 COMPILER V5.03, COMPILATION OF MODULE FLASH
OBJECT MODULE PLACED IN flash.OBJ
COMPILER INVOKED BY: F:\Keil\C166\BIN\C166.EXE flash.c MOD167 DEFINE(MONITOR) DEBUG CODE
SYMBOLS PAGELENGTH(18)

 stmt lvl     source

    1         #include <reg167.h>
    2
    3         void wait (void) { }                    /* wait function (empty) */
    4
    5         void main (void) {
    6  1
    7  1      unsigned int i;                         /* delay variable */

          (...)
```

Name of the output file (flash.obj)

Compilation process example: flashing LED

- Listing file *flash.lst*:

```
C166 COMPILER V5.03, FLASH
10/16/2004 17:52:41 PAGE 1


C166 COMPILER V5.03, COMPILATION OF MODULE FLASH
OBJECT MODULE PLACED IN flash.OBJ
COMPILER INVOKED BY: F:\Keil\C166\BIN\C166.EXE flash.c MOD167 DEFINE(MONITOR) DEBUG CODE
SYMBOLS PAGELENGTH(18)

 stmt lvl     source

    1         #include <reg167.h>
    2
    3         void wait (void) { }                    /* wait function (empty) */
    4
    5         void main (void) {
    6  1
    7  1      unsigned int i;                         /* delay variable */

  (...)
```

Compiler invocation command line

Compilation process example: flashing LED

- Listing file *flash.lst*:

```
C166 COMPILER V5.03, FLASH
10/16/2004 17:52:41 PAGE 1


C166 COMPILER V5.03, COMPILATION OF MODULE FLASH
OBJECT MODULE PLACED IN flash.OBJ
COMPILER INVOKED BY: F:\Keil\C166\BIN\C166.EXE flash.c MOD167 DEFINE(MONITOR) DEBUG CODE
SYMBOLS PAGELENGTH(18)

 stmt lvl     source

    1         #include <reg167.h>
    2
    3         void wait (void) { }                    /* wait function (empty) */
    4
    5         void main (void) {
    6  1
    7  1      unsigned int i;                         /* delay variable */

  (...)
```

Source code listing line numbers and source code

Compilation process example: flashing LED

```
C166 COMPILER V5.03, FLASH
10/16/2004 17:52:41 PAGE 3

ASSEMBLY LISTING OF GENERATED OBJECT CODE


            ; FUNCTION wait (BEGIN  RMASK = @0x8000)
                                    ; SOURCE LINE # 3
0000 CB00           RET
            ; FUNCTION wait (END    RMASK = @0x8000)

            ; FUNCTION main (BEGIN  RMASK = @0x4020)
                                    ; SOURCE LINE # 5
                                    ; SOURCE LINE # 10
0002 E6E1FF00       MOV       DP2,#0FFH
                                    ; SOURCE LINE # 11
0006 D180           EXTR      #01H
0008 E6E10000       MOV       ODP2,#00H
                                    ; SOURCE LINE # 13
```

Empty function *wait* is implemented as a simple return instruction *(RET*, machine code: *0xCB)*. A *zero-byte* has been inserted for *word alignment*

Compilation process example: flashing LED

```
C166 COMPILER V5.03, FLASH
10/16/2004 17:52:41 PAGE 3

ASSEMBLY LISTING OF GENERATED OBJECT CODE


            ; FUNCTION wait (BEGIN  RMASK = @0x8000)
                                    ; SOURCE LINE # 3
0000 CB00           RET
            ; FUNCTION wait (END    RMASK = @0x8000)

            ; FUNCTION main (BEGIN  RMASK = @0x4020)
                                    ; SOURCE LINE # 5
                                    ; SOURCE LINE # 10
0002 E6E1FF00       MOV       DP2,#0FFH
                                    ; SOURCE LINE # 11
0006 D180           EXTR      #01H
0008 E6E10000       MOV       ODP2,#00H
                                    ; SOURCE LINE # 13
```

Function *main* begins at relocatable address 0002 (following function *wait*).

First instruction: 'port 2 is output' *(DP2 = 0xFF)*

Compilation process example: flashing LED

```
000C            ?C0003:
                                    ; SOURCE LINE # 15
000C 76E00100       OR        P2,#01H
                                    ; SOURCE LINE # 16
0010 E005           MOV       R5,#00H
;---- Variable 'i' assigned to Register 'R5' ----
0012            ?C0008:
0012 BBF6           CALLR     wait
0014 86F50F27       CMPI1     R5,#0270FH
0018 8DFC           JMPR      cc_ULT,?C0008
001A            ?C0006:
                                    ; SOURCE LINE # 18
001A 66E0FEFF       AND       P2,#0FFFEH
                                    ; SOURCE LINE # 19
001E E005           MOV       R5,#00H
0020            ?C0013:
C166 COMPILER V5.03, FLASH
10/16/2004 17:52:41 PAGE 5

0020 BBEF           CALLR     wait
0022 86F50F27       CMPI1     R5,#0270FH
0026 8DFC           JMPR      cc_ULT,?C0013
0028            ?C0011:
                                    ; SOURCE LINE # 21
0028 0DF1           JMPR      cc_UC,?C0003
            ; FUNCTION main (END    RMASK = @0x4020)
```

Local variable, kept in R5 for fast access

Switch LED on and off *(set/ clear bit P2.0)*

while(1) { ... } loop, UnConditional jump to *?C0003*

Compilation process example: flashing LED

```
A166 MACRO ASSEMBLER  START167
10/16/2004 17:52:42 PAGE    38

                        618
                        619
                        620     ?C_RESET        PROC TASK C_STARTUP INTNO RESET = 0
                        621     ?C_STARTUP:     LABEL   NEAR
                        622
                        623
                        624     $IF (WATCHDOG = 0)
00000000 A55AA5A5       625                     DISWDT                  ; Disable watchdog
                                                                        ; timer
                        626     $ENDIF
                        627
                        635     $ENDIF
                        636
 0010                   637     BCON0L          SET     (_MTTC0 << 5) OR (_RWDC0 << 4)
 001E                   638     BCON0L          SET     BCON0L OR ((NOT _MCTC0) AND 0FH)
 001E                   639     BCON0L          SET     BCON0L AND (NOT (_RDYEN0 << 3))
 001E                   640     BCON0L          SET     BCON0L OR (_RDY_AS0 << 3)
```

Startup file *Start167.a66* initializes the principal system configuration registers and defines the stack; this code is called upon RESET and before *main*.

Compilation process example: flashing LED

```
(…)
                861     ;------------------------------------------------------------
                862     ;
                863     ; The following code is necessary to set RAM variables to
                864     ; 0 at start-up (RESET) of the C application program.
                865     ;
(…)
                975     ;------------------------------------------------------------
                976     ;
                977     ;  The following code is necessary, if the application
                978     ; program contains initialized variables at file level.
                979     ;
(…)
0000012A FA?????? E     1104                    JMP     FAR main
```

Start167 clears un-initialised variables (BSS) and moves initialised variables from EEPROM to RAM; then *main* is called (absolute address still unknown)

Compilation process example: flashing LED

```
"flash.obj",
"Start167.obj"
TO "flash"
RESERVE (8H-0BH, 0ACH-0AFH)
CLASSES (ICODE (0x4000-0x7FFF), NCODE (0x4000-0x7FFF),
FCONST (0x4000-0x7FFF), HCONST (0x4000-0x7FFF),
XCONST (0x4000-0x7FFF), NCONST (0x4000-0x7FFF),
NDATA (0x8000-0xBFFF), NDATA0 (0x8000-0xBFFF),
SDATA (0xE000-0xE7FF, 0xF600-0xFDFF), SDATA0 (0xE000-0xE7FF, 0xF600-0xFDFF),
IDATA (0xF600-0xFDFF), IDATA0 (0xF600-0xFDFF),
FDATA (0x8000-0xBFFF), FDATA0 (0x8000-0xBFFF),
HDATA (0x8000-0xBFFF), HDATA0 (0x8000-0xBFFF),
XDATA (0x8000-0xBFFF), XDATA0 (0x8000-0xBFFF))
 CINITTAB (0x4000-0x7FFF)
```

Linker script *flash.lnp* defines the linker options; objects *flash.obj* and *Start167.obj* are combined (linked) to output file *flash*. All absolute symbols are resolved and mapped to the specified addresses ranges (CONST = code, DATA = variables)

Compilation process example: flashing LED

```
L166 LINKER/LOCATER V5.05
10/16/2004  17:52:42  PAGE 1


L166 LINKER/LOCATER V4.25, INVOKED BY:
C:\PROGRAM FILES\KEIL\R423\C166\BIN\L166.EXE flash.obj, Start167.obj TO flash RESERVE (8H-0BH,
0ACH-0AFH) CLASSES (ICODE (0X4000-0X7FFF), NCODE (0X4000-0X7FFF), FCONST (0X4000-0X7FFF),
HCONST (0X4000-0X7FFF), XCONST (0X4000-0X7FFF), NCONST (0X4000-0X7FFF), NDATA (0X8000-0XBFFF),
NDATA0 (0X8000-0XBFFF), SDATA (0XE000-0XE7FF, 0XF600-0XFDFF), SDATA0 (0XE000-0XE7FF, 0XF600-
0XFDFF), IDATA (0XF600-0XFDFF), IDATA0 (0XF600-0XFDFF), FDATA (0X8000-0XBFFF), FDATA0 (0X8000-
0XBFFF), HDATA (0X8000-0XBFFF), HDATA0 (0X8000-0XBFFF), XDATA (0X8000-0XBFFF), XDATA0 (0X8000-
0XBFFF)) CINITTAB (0X4000-0X7FFF)

CPU TYPE:     C167 or derivative
CPU MODE:     SEGMENTED
MEMORY MODEL: SMALL
```

Linker output file *flash.m66* provides a summary of the link process (command line with all options, memory model, symbols, memory map, etc.); very useful for debugging!

Compilation process example: flashing LED

```
INTERRUPT PROCEDURES OF MODULE:  flash (FLASH)


INTERRUPT PROCEDURE               INT   INTERRUPT NAME
================================================
?C_RESET                            0   RESET


MEMORY MAP OF MODULE:  flash (FLASH)


START     STOP      LENGTH    TYPE  RTYP  ALIGN  TGR  GRP  COMB  CLASS    SECTION NAME
=============================================================================================
000000H   000003H   000004H   ---   ---   ---    ---  ---  ---   * INTVECTOR TABLE *
000008H   00000BH   000004H   ---   ---   ---    ---  ---  ---   * RESERVED MEMORY *
0000ACH   0000AFH   000004H   ---   ---   ---    ---  ---  ---   * RESERVED MEMORY *
004000H   004001H   000002H   XDATA REL   WORD   ---  ---  GLOB  ---      ?C_INITSEC
004002H   00412FH   00012EH   CODE  REL   WORD   ---  ---  PRIV  ICODE    ?C_STARTUP_CODE
004130H   004159H   00002AH   CODE  REL   WORD   ---    1  PUBL  NCODE    ?PR?FLASH
008000H   008FFFH   001000H   DATA  REL   WORD   ---    2  PUBL  NDATA    ?C_USERSTACK
00FA00H   00FBFFH   000200H   ---   ---   ---    ---  ---  ---   * SYSTEM STACK *
00FC00H   00FC1FH   000020H   DATA  ---   BYTE   ---  ---  ---   *REG*    ?C_MAINREGISTERS
```

flash.m66 includes information about the interrupt vectors and the memory map

Compilation process example: flashing LED

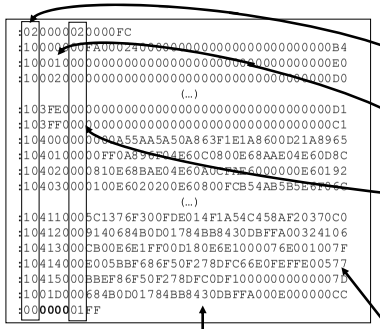
```
SYMBOL TABLE OF MODULE:  flash (FLASH)

      VALUE       TYPE      REP       LENGTH  TGR   SYMBOL NAME
      ========================================================
      004132H     GLOBAL    LABEL     ---     ---   main
      004130H     PUBLIC    LABEL     ---     ---   wait

      004130H     BLOCK     LVL=0     0002H   ---   wait
      ---         BLOCKEND  LVL=0     ---     ---

      004132H     BLOCK     LVL=0     0028H   ---   main
      004132H     BLOCK     LVL=1     0028H   ---
      000005H     SYMBOL    REG       ---     ---   i
      ---         BLOCKEND  LVL=1     ---     ---
      ---         BLOCKEND  LVL=0     ---     ---
      004002H     PUBLIC    LABEL     ---     ---   ?C_STARTUP
      008000H     PUBLIC    VAR       ---     ---   ?C_USRSTKBOT
      00FA00H     PUBLIC    CONST     ---     ---   ?C_SYSSTKBOT
      000000H     GLOBAL    INTNO     ---     ---   RESET
      000000H     SYMBOL    RBANK     ---     ---   ?C_MAINREGISTERS
      009000H     SYMBOL    VAR       ---     ---   ?C_USERSTKTOP
```

For every pair of curly brackets ({ }) the compiler introduces an internal label

flash.m66 includes a table of all symbols and their values, whether defined explicitly by the programmer or implicitly by the compiler

Compilation process example: flashing LED

File *flash.h86* is the linked executable in an EEPROM burner compatible format



Record length field (hex)

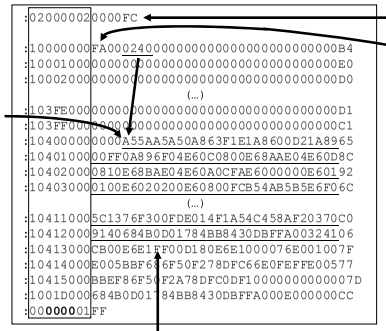
Address field (16-bit)

Record type field (00: data record, 01: end-of-file record, 02: 8086 segment address record

Data field

Check sum byte

Compilation process example: flashing LED



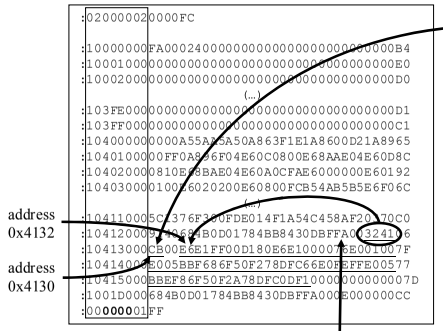
address 0x4002

segment address 0x0000

upon *RESET* the CPU diverts execution to address 0x0000 (RESET vector); here, this address contains JMPS 0x4002 (jump 'short' to the *startup code* found at absolute address 0x4002, note: little endian format)

instructions of the *startup code* (cf. slides MP2-28 and MP2-29)

Compilation process example: flashing LED



address 0x4132

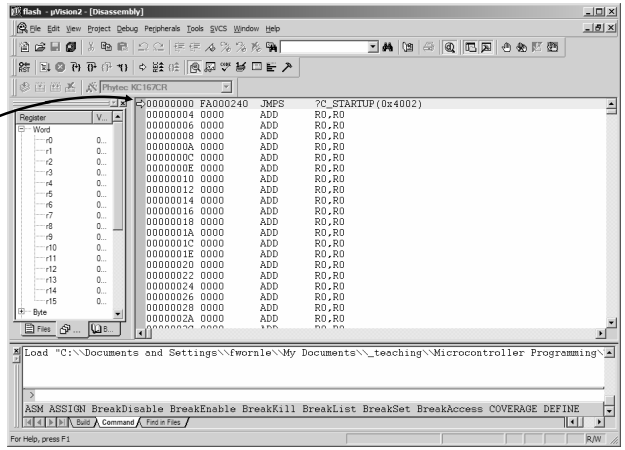
address 0x4130

absolute jump to 0x4132 (*main*, cf. slide MP2-29)

subroutine *wait* has been placed directly before *main* (address: 0x4130); byte CB corresponds to the RET instruction, the zero byte has been introduced to ensure that *main* begins on an even address (word alignment)

Debugging the LED program in *Simulation* mode

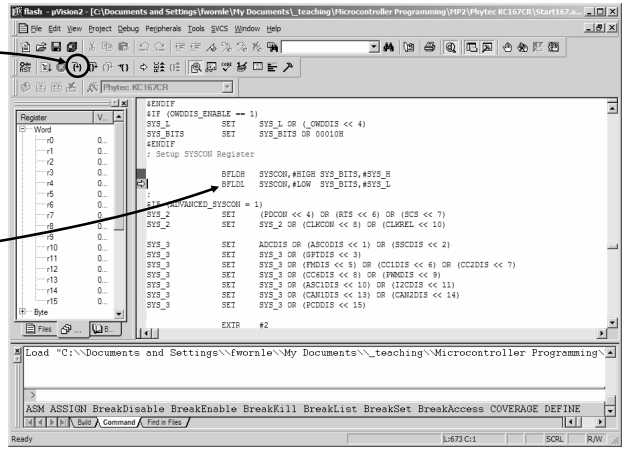
RESET: Fetch first instruction from address 0x0000



Debugging the LED program in *Simulation* mode

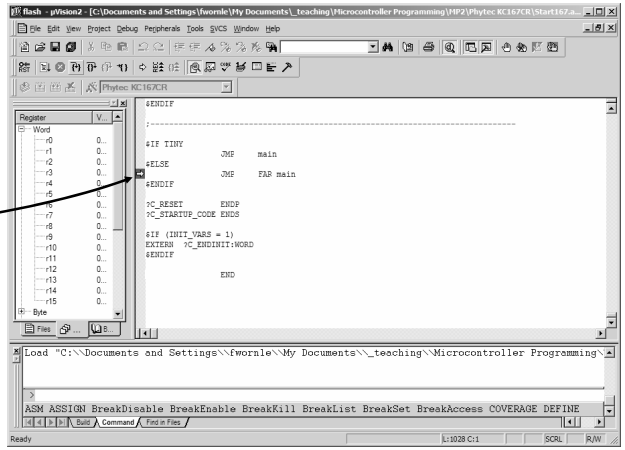
Single stepping through code…

Start-up code (initialize system registers)



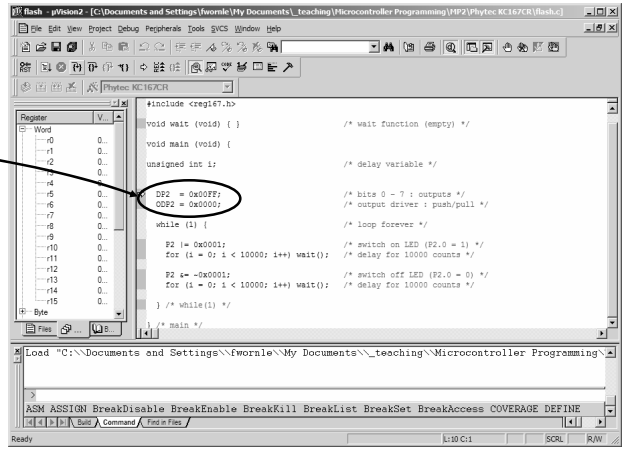
Debugging the LED program in *Simulation* mode

Start-up code concludes with a (far) jump to *main*



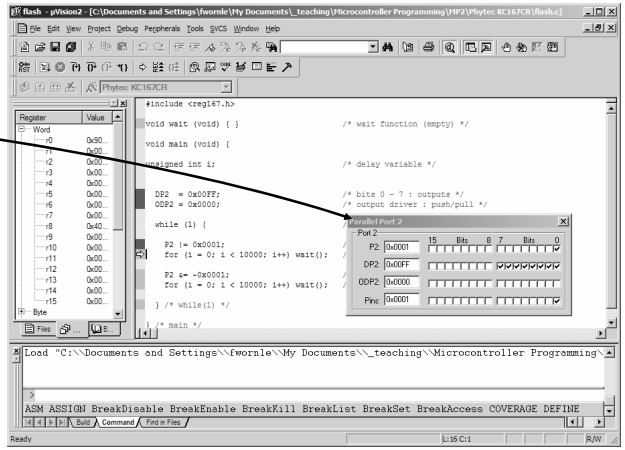
Debugging the LED program in *Simulation* mode

main: Initialize general purpose I/O port P2 (both, data direction and logic level)



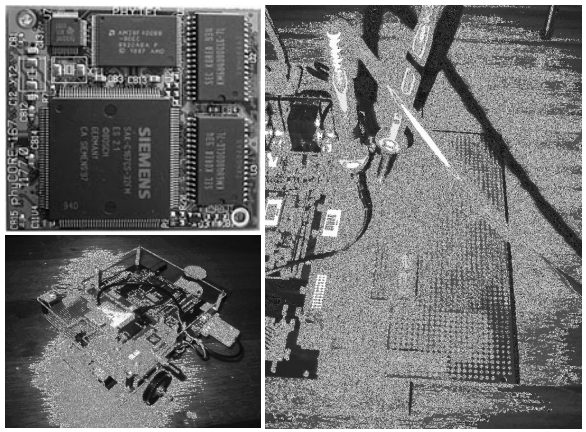
Debugging the LED program in *Simulation* mode

Simulator displays general purpose I/O ports; values can be modified manually



Debugging the LED program in *Monitor* mode

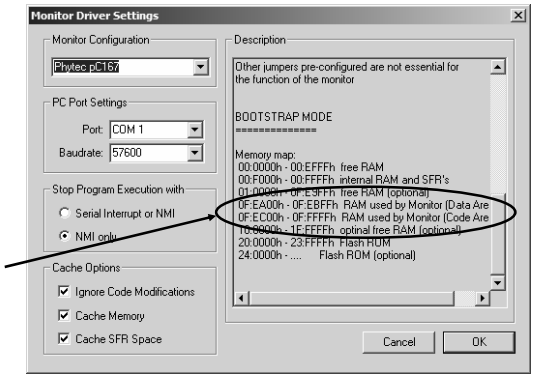
Alternatively: Monitor mode (download to target RAM – to allow setting of breakpoints, communica-tions via serial interface)



Debugging the LED program in *Monitor* mode

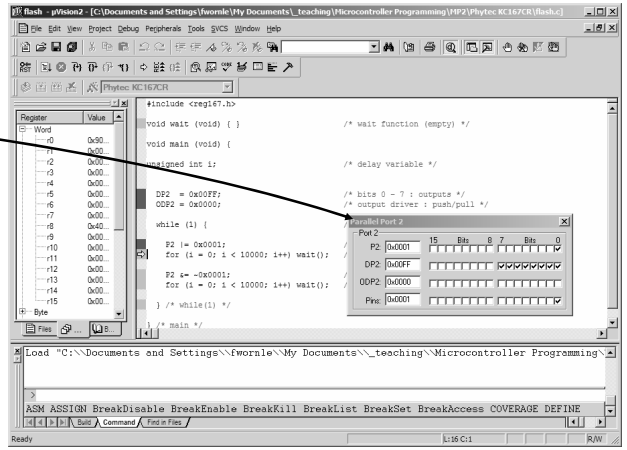
Monitor settings: Phytec board 'phyCore-167', communications through COM 1, 57600 bps, etc.

Monitor resides in the address space from 0xEA00 to 0xFFFF



Debugging the LED program in *Simulation* mode

Peripheral units (e.g. general purpose I/O ports, etc.) now represent the true state of the associated hardware



Summary – Build process