Electromechanical Timer Replacements

PIC16C5X Disassembler

Author: *Arsen Torbarina*
University of Electrical Engineering
Zagreb, Croatia

PROGRAM DEFINITION

DIS16 is an intelligent and easy-to-use disassembler for PIC16C5X microcontrollers. It produces a compact assembler source code from the binary data downloaded from a microcontroller's ROM and stored in a file.

FEATURES

- Supported processors: PIC16C54, PIC16C55, PIC16C56, PIC16C57, PIC16C58

  Processor can be automatically detected from the code.
- Supported file formats:
  - Intel® HEX Format (INHX8M, extension HEX)
  - Intel Split HEX Format (INHX8S, extensions HXL/HXH)
  - Raw binary file (BIN)
- Symbolic names for special-purpose registers (F00-F07) and flags in the STATUS register (F03) that enhances program readability.
- Intelligent program tracing that is used to:
  - Find and mark unreachable parts of a program
  - Detect multiple jumps (GOTO and CALL)
  - Tell apart program CODE from DATA tables
  - Determine required processor type, if none is specified
  - Detect and mark illegal instructions
- Generating useful comments automatically

Use of DIS16 disassembler requires an IBM PC/AT® or compatible computer, running MS-DOS® v4.1 or greater.

RUNNING DIS16

DIS16 can be invoked through the command line as follows:

```
DIS16 [options[+|-|argument]] filename
```

Where filename is a name of a file that contains program code, and options are used to pass some additional instructions to the disassembler.

The following file formats are supported:

- Intel HEX format (INHX8M, extension HEX)
- Intel split HEX format (INHX8S, extensions HXL/HXH)
- Raw binary file (BIN)

Note: It is not necessary to write the filename extension, since the program determines the correct extension by itself.

The program produces the following output files:

`filename.ASM`	Assembler listing, compatible with MPASM assembler;
`filename.LST`	Listing file suitable for printing and analyzing. Contains both HEX and symbolic code;
`filename.ERR`	Error file for details about the encountered errors and warnings

Note: All the previously existing files with the same names will be overwritten without any notice or warning.

OPTIONS

Option	Description	[Default]
/? or /h	Display help	
/c [+|-]	Set instructions case: +UPPER, -LOWER	[UPPER]
/p <procar>	Specify processor type, where <procar> is: 16C[54|55|56|57|58]	[autodetect]
/b = blsize	Set minimum block size that will be presented using assembler's FILL statement	[3]
/d:addrl [-addr2] [,addr3 [-addr4]…]	Force the code located between specified addresses to be interpreted as DATA (do not trace those addresses) (e.g. /d:20-2F, IE0, IF0-IFF)	
/1 [+|-]	Enable or disable listing file	[On]
/e [+|-]	Enable or disable error file	[On]
/q [+|-]	Enable or disable quiet mode (suppresses screen output)	[Off]

ENHANCING THE PROGRAM READABLITY

There are several mechanisms that DIS16 uses to enhance the program readability:

Symbolic names

Disassembler generates a table of used symbols, such as file registers and STATUS register bits, and assigns them symbolic names as shown below:

1. Destination at “Byte-oriented file register operations”

```
        W         EQU    0         ;Destination is W
        F         EQU    1         ;Destination is f
```

2. File register names

```
        INDF      EQU    0x00     ;Indirect data addressing
        RTCC      EQU    0x01     ;Real Time Clock/Counter
        PC        EQU    0x02     ;Program Counter
        STATUS    EQU    0x03     ;Status register
        FSR       EQU    0x04     ;File Select Register
        PORTA     EQU    0x05     ;Port A address
        PORTB     EQU    0x06     ;Port B address
        PORTC     EQU    0x07     ;Port C address [16C55,57]
or      F07       EQU    0x07     ;General purpose [16C54,56,58]
        F08       EQU    0x08     ;General purpose register
                  ,,,,,
        F1F       EQU    0xlF     ;General purpose register
```

3. STATUS register bit names

```
        C         EQU    0        ;Carry bit
        DC        EQU    1        ;Digit Carry bit
        Z         EQU    2        ;Zero bit
        NOT_PD    EQU    3        ;NOT Power Down bit
        NOT_TO    EQU    4        ;NOT Time-Out bit
        PA0       EQU    5        ;Page-bit 0
        PA1       EQU    6        ;Page-bit 1
        PA2       EQU    7        ;Page-bit 2 (not used)
```

Generating Comments

There are three groups of comments that can be generated automatically by DIS16:

1. Warnings:

   WARNING – Presumed value

   Next to 'PROCESSOR xxxxx' statement, where no processor type is given, so the processor type had to be automatically determined. It always gives the cheapest processor required.

   WARNING – PORTC not supported by this processor

   Occurs when 'TRIS PORTC' is encountered in the code for a PIC16C54, PIC16C56 or 1PIC6C58 processor.

2. Errors:

   ERROR – More than 12 bits

   Occurs if an instruction contains more than 12 bits, since PIC16C5X instructions have only 12 bits.

   ERROR – Illegal instruction

   Occurs if a code, *not* previously defined as DATA (see “Specifying DATA Fields Manually”) does not correspond to any of the defined instructions.

   To prevent this error, all the addresses where it occurs define as DATA using the /d option.

3. Remarks:

   Addresses forced to be interpreted as DATA.

   Below this, follows a list of the addresses that have been defined as DATA addresses. See also “Specifying DATA Fields Manually.”

   Unreachable code

   The code below (to the first blank line) most probably will not be executed. However, it doesn't always have to be true, since DIS16 will not trace jumps made by modifying INDF and PC registers (e.g., MOVF PC). See also ‘GOTO and CALL statements.’

   Could be…

   List of credible jump destinations, written next to a GOTO or CALL instruction. Occurs only if there was multiple page addressing detected and there are more possible destinations (e.g., 'Could be L020 L220 L620'). See also "GOTO and CALL statements".

Compacting Blocks with FILL Statement (/b Option)

If DIS16 encounters a byte that fills a block larger or equal to a given value, it will use a FILL statement to present it in the assembler code. That value is by default set to 3, but you can change it using the /b option (e.g. /b=10 will present all the blocks larger or equal to 10 bytes with a FILL statement). This value must be at least 2.

NOP (or $000 code) blocks will ***not*** be presented by an FILL statement and will be treated as unused space. An ORG instruction will be used instead to set address of the code that comes afterwards.

If a byte that repeats is an instruction without a comma inside (e.g., SLEEP or CLRW f), then the following syntax will be used:

```
FILL (instruction),repeats
```

But, if the instruction contains a comma (e.g., MOVF f,d), then just its code will be written, instead of the instruction itself, and the instruction will be written as comment, next to the FILL statement:

```
FILL 0x330,0x4 ; RRF Fl0,F
```

It had to be done that way, because the MPASM assembler wouldn't accept it otherwise. See Example 1 for additional information.

EXAMPLE 1: DISASSEMBLING ORIGINAL SOURCE CODE

Let's take an assembler source like this one:

```
; --------- original source code ---------
           processor 16c54
           org       OxOOO
start      rrf       OxlO,1
           rrf       OxlO,1
           rrf       OxlO,1
           rrf       OxlO,1                  ; Divide F10 by 16
           movf      OxlO,O
           btfsc     Ox03,2                  ; Skip if ZERO
           goto      notzero
           goto      zero
notzero    incf      Oxll, 1                 ; Increment F11 if not zero
           goto      start
           org       0x100
zero       incf      0x12,1                  ; Increment F12 if zero
           movef     0x11,0                  ; Copy F11 to W
           movewf    0x10                    ; Copy W to F10
           goto      start                   ; and do it again.
           fill      (clrf 11),0x1f0-$       ; Fill to Oxlef with 'CLRF 1'
           end
```

If you compiled it, and disassembled it again using DIS16, you would get the following output:

```
; --------- Generated by DIS16 v1.00.00b Disassembler --------

         PROCESSOR   16C54                   ; WARNING - Presumed value!

W          EQU       0
F          EQU       1
z          EQU       2
STATUS     EQU       0x03
F10        EQU       0x10
F11        EQU       0x11
F12        EQU       0x12

           ORG       OxOOO
L000       FILL      0x330,0x4; RRF F10,F
           MOVF      F10,W
           BTFSC     STATUS,Z
           GOTO      L008
           GOTO      L100

L008       INCF      Fll,F
           GOTO      L000

           ORG       OxlOO
L100       INCF      F12,F
           MOVF      Fll,W
           MOVWF     F10
           GOTO      L000

; Unreachable code

L104       FILL      (CLRF F11),0xEC

END
```

TRACING THE PROGRAM

There is an intelligent trace algorithm applied on the input code, that is used to:

- Find and mark unreachable parts of a program
- Determine possible destination pages in GOTO and CALL
- Determine required processor type, if none is specified
- Detect and mark illegal instructions

After loading, DIS16 starts to follow instruction by instruction of the loaded program, starting from the highest available address (for 16C57 it will be $7FF). Every time it encounters a “skip if…” branch statement it branches and continues from the both possible addresses.

Although the tracer follows all the jumps, it cannot get stuck into an endless loop, since each address can be visited only up to 5 times. When it encounters an address which has been executed more than that, the tracer closes that branch and returns to the point BEFORE it entered the branch.

GOTO and CALL Statements

When a `GOTO` or `CALL` statement occurs, DIS16 disassembler will test which pages that instruction may point to. It is done by testing whether in the previous steps the Indirect Data

Addressing register (INDF) or STATUS register (that contains page select bits) have been modified. If so, then it will be tested whether the possible destination addresses on all the available pages contain something other than a NOP J4F instruction. So they will be considered as credible destinations for jump and thus written in the comment next to the `GOTO` or `CALL` instruction.

Note:	This algorithm is not 100% reliable, since there always could be some exceptions (e.g., if a `GOTO` deliberately points to a `NOP` statement), but in most cases it will give satisfying results. Furthermore, DIS16 will not consider jumps made by writing directly to PC (F02) or INDF (F00).

For example, DIS16 could produce an output like this one:

```
STATUS EQU   0x03


 L000  ORG   OxOOO
       ....          ;(some code)
       MOVWF STATUS
       ,,,,
       CALL  0x020  ;Could be L020 L220
       CALL  0x020  ;Could be L030 L630
       GOTO  0x030
 L020  ,,,,
       ,,,,
 L030  ....
       ....
       ORG   0x220
 L220  ....
       RETLW 0
       ORG   0x630
 L630  ....
       END
```

You can see that in the `CALL 0x020` instruction there are two addresses that are considered as destinations, since both of them point to a code other than `NOP`. There is the same situation with the `CALL 0x030` instruction.

However, in the `GOTO L000` statement, there is only one possibility (and that is L000), since addresses 0x200, 0x400, and 0x600 contain no code.

Unreachable Parts

Any code that was never reached during the tracing procedure will be marked with an “Unreachable code” remark. However, it doesn't always mean that the code cannot be reached (see the Note in the previous paragraph).

Specifying DATA Fields Manually

Since the tracer treats all the code by default as executable instructions, sometimes you will have to tell to the disassembler which addresses are to be treated as DATA and not to be traced.

EXAMPLE 2: MANUALLY SPECIFYING DATA FIELDS USING TEST.HEX

If you using the following file (test.hex)

```
:08000000000A44004100540015
:020008004100B5
:00000001FF
```

and you try to disassemble it by typing only:

```
DIS16 test.bin
```

you will get the following output:

```
        ORG   0x000
L000    GOTO  L000
; Unreachable code
L001    DW    0x044     ;ERROR - Illegal instruction
        DW    0x041     ;ERROR - Illegal instruction
        DW    0x054     ;ERROR - Illegal instruction
        DW    0x041     ;ERROR - Illegal instruction
END
```

But if you tell the disassembler that the code between addresses $001 and $004 is DATA section by typing:

```
DIS16 test.hex /d:1-4
```

the following output is produced:

```
; Addresses forced to be interpreted as DATA:
; 0x001 - 0x004

ORG     0x000
L000    GOTO  L000
DW      0x044           ; 'D'
DW      0x041           ; 'A'
DW      0x054           ; 'T'
DW      0x041           ; 'A'

END
```

You can also specify up to 100 different DATA areas, such as in:

```
DIS16 prog.hex /d:100-lFF,210,250-25F
```

See also "Compacting blocks with FILL statement'

Note: Data blocks will never be marked with the "Unreachable code" comment.

Determining Processor Type

Although advisable, it is not always necessary to specify the processor type, since DIS16 can determine by itself the cheapest processor required for the given program. The choice is made regarding the program length and the used ports.

For example, if it encounters a TRIS 7 (initialize Port C) instruction, it is obvious that it requires a 16C55 or 16C57 processor. Besides, if the program contains up to 1K, the suggested processor will be 16C55.

Illegal Instructions

Any encountered illegal instruction (e.g., $001, $041-$04F) that was not previously defined as DATA (see “Specifying DATA Fields Manually”) will be presented using a `DW` statement.

Also there will be an “ERROR - Illegal instruction” comment added next to the `DW` statement and to the ERR file.

WORLDWIDE SALES & SERVICE

AMERICAS

Corporate Office
Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 602-786-7200 Fax: 602-786-7277
Technical Support: 602 786-7627
Web: http://www.microchip.com

Atlanta
Microchip Technology Inc.
500 Sugar Mill Road, Suite 200B
Atlanta, GA 30350
Tel: 770-640-0034 Fax: 770-640-0307

Boston
Microchip Technology Inc.
5 Mount Royal Avenue
Marlborough, MA 01752
Tel: 508-480-9990 Fax: 508-480-8575

Chicago
Microchip Technology Inc.
333 Pierce Road, Suite 180
Itasca, IL 60143
Tel: 630-285-0071 Fax: 630-285-0075

Dallas
Microchip Technology Inc.
14651 Dallas Parkway, Suite 816
Dallas, TX 75240-8809
Tel: 972-991-7177 Fax: 972-991-8588

Dayton
Microchip Technology Inc.
Two Prestige Place, Suite 150
Miamisburg, OH 45342
Tel: 937-291-1654 Fax: 937-291-9175

Los Angeles
Microchip Technology Inc.
18201 Von Karman, Suite 1090
Irvine, CA 92612
Tel: 714-263-1888 Fax: 714-263-1338

New York
Microchip Technology Inc.
150 Motor Parkway, Suite 416
Hauppauge, NY 11788
Tel: 516-273-5305 Fax: 516-273-5335

San Jose
Microchip Technology Inc.
2107 North First Street, Suite 590
San Jose, CA 95131
Tel: 408-436-7950 Fax: 408-436-7955

Toronto
Microchip Technology Inc.
5925 Airport Road, Suite 200
Mississauga, Ontario L4V 1W1, Canada
Tel: 905-405-6279 Fax: 905-405-6253

ASIA/PACIFIC

Hong Kong
Microchip Asia Pacific
RM 3801B, Tower Two
Metroplaza
223 Hing Fong Road
Kwai Fong, N.T., Hong Kong
Tel: 852-2-401-1200 Fax: 852-2-401-3431

India
Microchip Technology Inc.
India Liaison Office
No. 6, Legacy, Convent Road
Bangalore 560 025, India
Tel: 91-80-229-4036 Fax: 91-80-559-9840

Korea
Microchip Technology Korea
168-1, Youngbo Bldg. 3 Floor
Samsung-Dong, Kangnam-Ku
Seoul, Korea
Tel: 82-2-554-7200 Fax: 82-2-558-5934

Shanghai
Microchip Technology
RM 406 Shanghai Golden Bridge Bldg.
2077 Yan'an Road West, Hong Qiao District
Shanghai, PRC 200335
Tel: 86-21-6275-5700
Fax: 86 21-6275-5060

Singapore
Microchip Technology Taiwan
Singapore Branch
200 Middle Road
#10-03 Prime Centre
Singapore 188980
Tel: 65-334-8870 Fax: 65-334-8850

Taiwan, R.O.C
Microchip Technology Taiwan
10F-1C 207
Tung Hua North Road
Taipei, Taiwan, ROC
Tel: 886 2-717-7175 Fax: 886-2-545-0139

EUROPE

United Kingdom
Arizona Microchip Technology Ltd.
Unit 6, The Courtyard
Meadow Bank, Furlong Road
Bourne End, Buckinghamshire SL8 5AJ
Tel: 44-1628-851077 Fax: 44-1628-850259

France
Arizona Microchip Technology SARL
Zone Industrielle de la Bonde
2 Rue du Buisson aux Fraises
91300 Massy, France
Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79

Germany
Arizona Microchip Technology GmbH
Gustav-Heinemann-Ring 125
D-81739 Müchen, Germany
Tel: 49-89-627-144 0 Fax: 49-89-627-144-44

Italy
Arizona Microchip Technology SRL
Centro Direzionale Colleoni
Palazzo Taurus 1 V. Le Colleoni 1
20041 Agrate Brianza
Milan, Italy
Tel: 39-39-6899939 Fax: 39-39-6899883

JAPAN

Microchip Technology Intl. Inc.
Benex S-1 6F
3-18-20, Shinyokohama
Kohoku-Ku, Yokohama-shi
Kanagawa 222 Japan
Tel: 81-45-471- 6166 Fax: 81-45-471-6122

7/29/97