ADR2 Digirule 2U Cross-Assembler User Manual

8 January 2021

Introduction

ADR2 is a traditional style command line cross-assembler for the Digirule 2U. Its purpose is to translate assembly language program source into machine code for downloading to the Digirule 2U. Versions are available for Linux, Mac OS and Windows.

The assembler has the following features:

- Supports the entire Digirule 2U instruction set
- Supports directives for control of the assembly and listing process
- Supports definition of symbolic names to represent memory addresses and constants
- Supports symbolic expressions with a comprehensive set of operators
- Supports pre-defined symbols for commonly-used Digirule 2U resources
- Supports numbers represented in binary, decimal, hexadecimal and character notation
- Produces a formatted listing file which includes location counter, machine code, source line, symbol table and RAM usage information
- Two-pass assembly
- Directly produces machine code in Intel Hex format for download to the Digirule 2U

Invoking the Assembler

The assembler is invoked using the following general syntax:

```
adr2 [options] <sourcefile >hexfile
```

The assembler reads the source input from standard input by default, so it may be redirected from a file, device or another program as desired. The assembler writes the hex output to standard output by default, so it may be redirected to a file, device or another program as desired.

The assembler supports the following command-line options:

Option	Function
`--quiet`	suppress banner and completion info
`--nolist`	suppress listing file creation
`--list [path]`	specify alternate listing file pathname (default is `adr2.lst`)
`--help`	display help information

Here are a few examples of how to invoke the assembler:

listing to `adr2.lst`, hex to `sample.hex`

```
adr2 < sample.asm > sample.hex
```

no listing, hex to `sample.hex`

```
adr2 --nolist < sample.asm > sample.hex
```

listing to `sample.lst`, hex to `sample.hex`

```
adr2 --list sample.lst < sample.asm > sample.hex
```

listing to `sample.lst`, hex downloaded to Digirule 2U^*^ (Linux)

```
stty -F /dev/ttyUSB0 9600 cs8 -parenb -cstopb
adr2 --list sample.lst < sample.asm > /dev/ttyUSB0
```

listing to `sample.lst`, hex downloaded to Digirule 2U^*^ (via `dldr2` download utility, Mac)

```
adr2 --list sample.lst < sample.asm | dldr2 /dev/tty.usbserial-FTxxxxxx
```

listing to `sample.lst`, hex downloaded to Digirule 2U^*^ (Windows)

```
mode COM3 baud=9600 data=8 parity=n stop=1
adr2 --list sample.lst < sample.asm > COM3
```

> Tip: For Windows, if the port device name is `COM10` or greater, it must be specified as `\\.\COMnn`

> ^*^This example requires the Digirule 2U to be in Comm Load Mode prior to execution. Please refer to the *Digirule 2U User Manual* for details on setting the mode.

The assembler returns a value to the operating system indicating if errors or warnings were encountered during assembly.

Return Value	Meaning
0	Success
1	Warning(s)
2	Error(s)
3	Fatal error

Assembler Input

The assembler recognizes the following characters for source input. All other characters are ignored.

Category	Characters
Whitespace	space, horizontal tab
Letters	A..Z, a..z
Digits	0..9
Punctuation	`! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~`
Line Feed	0x0A

Source File Format

A program is written as one or more source files. A source file consists of source lines, each with a specific format. The main source file typically starts with an `ORG` directive to set the initial location counter, and an `END` directive to mark the end of assembly (and optionally specify the program entry point).

> Tip: If the `ORG` directive is omitted, the location counter defaults to 0 at the beginning of assembly.
>
> Tip: If the `END` directive is omitted, assembly stops at the end of the file, and the program entry point defaults to 0.

Source Line Format

Each source line has a specific format consisting of up to four fields. Fields are separated by whitespace (one or more horizontal tab or space characters), so fields may not contain whitespace. The exceptions are inside quoted character constant operands and comments. The fields, in order from left to right, are:

- Label Field

- Operation Field
- Operand(s) Field
- Comment Field

```
loop  copyrr  src,dst   save the data
```

The maximum length of a line is 255 characters. Any characters beyond this limit are ignored.

Label Field

The label field optionally contains a user-defined label. If a label is present, its first character must be an underscore or a letter. Subsequent characters may be underscores, letters or digits. The maximum length of a label is 255 characters. Labels *are* case-sensitive, i.e. `LOOP` and `loop` are distinct. A label will assume the current value of the location counter unless the operation field is the `EQU` directive, in which case the label will assume the value of the expression following the directive.

Operation Field

The operation field optionally contains an instruction mnemonic or an assembler directive. If no instruction is present, no machine code will be emitted for that line. Instruction mnemonics and directives *are not* case sensitive.

Operand Field

The operand field contains any operand(s) required by the operation field instruction or directive. If multiple operands are required, **they must be separated by commas**. It is an error to have too few or too many operands for a given operation.

Comment Field

The comment field optionally contains human readable text typically used to document the program. The assembler ignores the contents of this field, except that it is written to the listing file.

> Tip: If a comment is desired on a line lacking an operation or operand, the comment must begin with a semicolon `;`.
>
> Tip: A line that begins with an asterisk `*` or a semicolon `;` is treated as a comment and ignored.

Directives

The assembler supports the following directives:

Directive	Function
`BYTE` *exp* [,...]	Defines one or more bytes starting at the current location counter and initializes them to *exp* [,...]
`END` [*exp*]	Optionally marks the end of the assembly source file. Any lines that follow will be ignored. The optional *exp* specifies the program entry point, which will be automatically loaded into the Digirule 2U Program Counter upon download.
label `EQU` *exp*	Assigns the new symbol *label* the value *exp*. This is the only way to assign a value other than the location counter to a label. *exp* must not reference a forward or undefined symbol.
`INCLUDE` *file*	Include the contents of another source file. Included files may themselves include other files, up to a system-dependent depth. The remainder of the including source file will be processed after the entire included source file is processed.
`LIST`	Include the following lines in the listing file.
`NOLIST`	Exclude the following lines from the listing file.
`NOPAGE`	Suppress page generation in the listing file. Once specified, it is not possible to return to automatic page generation.
`ORG` *exp*	Changes the location counter to *exp*. Subsequent lines are assigned memory locations starting at the new location counter value. *exp* must not be a value lower than the current location counter.
`PAGE`	Starts a new page in the listing file.
`SPACE` *exp*	Advances the location counter to reserve *exp* bytes of memory. **The memory is not initialized in any way.** This directive, preceded by a label, is typically used to create an uninitialized buffer.

Constants

Numeric constants may be expressed in binary, decimal, hexadecimal or character notation.

Radix	Example
Binary	`0b01000001`
Decimal	`65`
Hexadecimal	`0x41`
Character	`'A'`
Location counter	`*`

> Tip: To embed a quote character within a quoted character or string constant, use *two* adjacent quotes, e.g. `str  byte  'Brent''s Digirule 2U',0`

Expression Operators

The following assembly-time operators are supported, grouped by equal precedence, in decreasing order of precedence:

Operator	Function
`(`, `)`	Change the order of evaluation
`+`, `-`, `~`	Unary plus, unary minus, unary not
`<<`, `>>`, `<<<`, `>>>`	Shift left, shift right, rotate left, rotate right
`&`	And
`^`	Exclusive-or
`|`	Or
`**`	Power
`*`, `/`, `%`	Multiply, divide, modulo (remainder)
`+`, `-`	Add, subtract

Symbols are stored, and symbolic expressions are evaluated, with 16-bit precision.

Predefined Symbols

The following symbols are conveniently predefined by the assembler:

Symbol	Resource	Value
`_z`	Status register *Zero* bit	0x00
`_c`	Status register *Carry* bit	0x01
`_sar`	Status register *Show Address Register* bit	0x02
`_sr`	Status register	0xFC
`_br`	Button register	0xFD
`_ar`	Address LEDs register	0xFE
`_dr`	Data LEDs register	0xFF

EXAMPLE

```
ram_end   EQU _dr
code    EQU (ram_end+1)*3/4
    ORG code
    COPYLR  1<<7|1<<3,var+2
```

Assembler Output

Hex File

The hex file contains the assembler-generated machine code in Intel Hex format, suitable for download to the Digirule 2U. Each line is a record, consisting of the following fields:

- start character (`:`)
- data field byte count (two digits)
- starting memory address (four digits)
- record type (`00` for data record, `01` for end-of-file record)
- data field (up to 32 digits)
- checksum (two digits)

Record type `00` is a data record containing data to initialize Digirule 2U memory. Record type `01` is an end-of-file record containing the program entry point address. **Only those memory locations containing instructions or data generated with the `BYTE` directive are included in the hex file and written to Digirule 2U memory during download. Other memory locations will remain unchanged.** This permits downloading multiple programs to different addresses if desired.

```
:10000000032402FC291C0AFDFD2500FC27041EF028
:10001000291C0AFDFD2600FC271027042301FC25CE
:1000200000F0273709F1110F07F109FF110007FF51
:1000300009FE110007FE2B09F1130F07F109FF1349
:0A0040000007FF09FE130007FE2B66
:0100F000000F
:00000001FF
```

Listing File

The listing file combines the source code and the corresponding assembler-generated machine code for easy cross reference. It is an invaluable source of information when debugging a program. From left to right, a typical listing line contains:

- Location counter
- Generated machine code (left justified) or expression value (right justified)
- Source file line number
- Source file line contents

```
0004  2A 1C         8.  loop    call    count
                    9.
0006  0A FD FD     10.          copyrr  _br,_br
0009  26 00 FC     11.          btstsc  _z,_sr
000C  28 04        12.          jump    loop
```

Near the end of the listing file is a dump of the internal symbol table, showing every defined symbol, its associated value, and the source line number at which it was defined.

```
Line   Val   Symbol
-----  ----  --------------------------------------------------
    0  00FE  _ar
    0  00FD  _br
    0  0001  _c
    0  00FF  _dr
    0  0002  _sar
    0  00FC  _sr
    0  0000  _z
   61  00F1  byte0
   67  00FF  byte1
   64  00FE  byte2
   24  001C  count
   59  00F0  dir
   43  0037  down
    8  0004  loop
   16  0010  loop2
    5  0000  start
    1  000F  step
```

```
   29  0024  up
```

At the far end is a RAM usage table, showing which memory locations will be written when the corresponding hex file is downloaded to the Digirule 2U.

```
RAM Usage
---------------------
0000  XXXXXXXXXXXXXXXX
0010  XXXXXXXXXXXXXXXX
0020  XXXXXXXXXXXXXXXX
0030  XXXXXXXXXXXXXXXX
0040  XXXXXXXXXX......
0050  ................
0060  ................
0070  ................
0080  ................
0090  ................
00A0  ................
00B0  ................
00C0  ................
00D0  ................
00E0  ................
00F0  X...........
```

Sample Source File

```
step    equ     15

        org     0

start   initsp
        bset    _sar,_sr

loop    call    count

        copyrr  _br,_br
        btstsc  _z,_sr
        jump    loop

        incr    dir

loop2   call    count

        copyrr  _br,_br
        btstss  _z,_sr
        jump    loop2
```

```
        jump    loop

count   bclr    _c,_sr

        btstsc  0,dir
        jump    down

up      copyra  byte0
        addla   step
        copyar  byte0

        copyra  byte1
        addla   0
        copyar  byte1

        copyra  byte2
        addla   0
        copyar  byte2

        return

down    copyra  byte0
        subla   step
        copyar  byte0

        copyra  byte1
        subla   0
        copyar  byte1

        copyra  byte2
        subla   0
        copyar  byte2

        return

        org     0xF0

dir     byte    0

byte0   space   1

        org     _ar
byte2   space   1

        org     _dr
byte1   space   1

        end     start
```

Sample Listing File

```
Digirule 2U Cross-Assembler | Wed Sep 23 10:55:53 2020 | Page 1

          000F      1.  step    equ     15
                    2.
0000                3.          org     0
                    4.
0000  03            5.  start   initsp
0001  24 02 FC      6.          bset    _sar,_sr
                    7.
0004  2A 1C         8.  loop    call    count
                    9.
0006  0A FD FD     10.          copyrr  _br,_br
0009  26 00 FC     11.          btstsc  _z,_sr
000C  28 04        12.          jump    loop
                   13.
000E  1E F0        14.          incr    dir
                   15.
0010  2A 1C        16.  loop2   call    count
                   17.
0012  0A FD FD     18.          copyrr  _br,_br
0015  27 00 FC     19.          btstss  _z,_sr
0018  28 10        20.          jump    loop2
                   21.
001A  28 04        22.          jump    loop
                   23.
001C  23 01 FC     24.  count   bclr    _c,_sr
                   25.
001F  26 00 F0     26.          btstsc  0,dir
0022  28 37        27.          jump    down
                   28.
0024  09 F1        29.  up      copyra  byte0
0026  11 0F        30.          addla   step
0028  07 F1        31.          copyar  byte0
                   32.
002A  09 FF        33.          copyra  byte1
002C  11 00        34.          addla   0
002E  07 FF        35.          copyar  byte1
                   36.
0030  09 FE        37.          copyra  byte2
0032  11 00        38.          addla   0
0034  07 FE        39.          copyar  byte2
                   40.
0036  2C           41.          return
                   42.
0037  09 F1        43.  down    copyra  byte0
```

```
0039  13 0F        44.          subla   step
003B  07 F1        45.          copyar  byte0
                   46.
003D  09 FF        47.          copyra  byte1
003F  13 00        48.          subla   0
0041  07 FF        49.          copyar  byte1
                   50.
0043  09 FE        51.          copyra  byte2
0045  13 00        52.          subla   0
0047  07 FE        53.          copyar  byte2
                   54.
0049  2C           55.          return
                   56.
00F0               57.          org     0xF0
                   58.
Digirule 2U Cross-Assembler | Wed Sep 23 10:55:53 2020 | Page 2

00F0  00           59.  dir     byte    0
                   60.
00F1               61.  byte0   space   1
                   62.
00FE               63.          org     _ar
00FE               64.  byte2   space   1
                   65.
00FF               66.          org     _dr
00FF               67.  byte1   space   1
                   68.
0100      0000     69.          end     start

Assembly complete
0 errors
0 warnings


Line   Val   Symbol
-----  ----  --------------------------------------------------
    0  00FE  _ar
    0  00FD  _br
    0  0001  _c
    0  00FF  _dr
    0  0002  _sar
    0  00FC  _sr
    0  0000  _z
   61  00F1  byte0
   67  00FF  byte1
   64  00FE  byte2
   24  001C  count
   59  00F0  dir
   43  0037  down
```

```
    8  0004  loop
   16  0010  loop2
    5  0000  start
    1  000F  step
   29  0024  up


RAM Usage
----------------------
0000  XXXXXXXXXXXXXXXX
0010  XXXXXXXXXXXXXXXX
0020  XXXXXXXXXXXXXXXX
0030  XXXXXXXXXXXXXXXX
0040  XXXXXXXXXX......
0050  ................
0060  ................
0070  ................
0080  ................
0090  ................
00A0  ................
00B0  ................
00C0  ................
00D0  ................
00E0  ................
00F0  X...........
```