NAME

dxa - 6502/R65C02 disassembler

SYNOPSIS

dxa [*OPTION*]... *FILE*

DESCRIPTION

dxa is the semi-official disassembler option for the **xa(1)** package, a weakly patched version of Marko Mäkelä's **d65** disassembler that generates output similar to the de facto coding conventions used for **xa(1).** The package is designed to intelligently(?) scan arbitrary code and (with hints) can identify the difference between data and valid machine code, generating a sane looking, "perfect" disassembly with data and code portions.

Perfect, in this case, means that you can take what **dxa** spits out and feed it right back into **xa(1)**, and get the exact same object file you started with, even if sometimes **dxa** can't identify everything correctly. With a few extra options, you can tease and twist the output to generate something not quite so parseable, or even more like true assembler source.

OPTIONS

For historical and compatibility reasons, the long options (--) only exist if **dxa** were compiled with LONG_OPTIONS enabled in options.h.

--datablock xxxx-yyyy

-b xxxx-yyyy
Defines the memory range xxxx to yyyy (hexadecimal, inclusive) to be a data block. The memory range can be further specified:

* If the range is preceded by ! (an exclamation point), such as **!c000-cfff**, then it is further defined to be a data block with no vectors in it either.

* If the range is preceded by ? (a question mark), then it is further defined to be a data block that is completely unused and therefore no valid routine may contain instructions whose parameter lie in this range. Useful for providing enhanced protection against misinterpreting data to be program code, but be careful, or some code may be listed as data. For instance, the Commodore 64 firmware uses the base address $CFFF when initializing the video chip, and the BASIC interpreter uses the addresses $9FEA and $9FEB while loading the instruction pointers. In addition to this, there are a number of BIT instructions used for skipping the next instruction. Thus, you must allow addresses like $1A9, $2A9 and so on.

--datablocks filename

-B filename
Reads data blocks from file **filename** as if they had been specified on the command line, one per line (such as xxxx-yyyy, ?xxxx-yyyy, etc.).

--labels filename

-l filename
Causes label names to be read from file **filename.** This file format is the same as the labelfile/symbol table file generated by **xa(1)** with the **-l** option. The **-l** was chosen on purpose for consistency with **xa(1).**

--routine xxxx

-r xxxx
Specifies an address (in hexadecimal) that is declared to be a valid routine. **It is strongly recommended** that you specify the initial execution address as a routine. For example, for a Commodore 64 binary with a **SYS 2064** header, add **-r0810** so that disassembly starts at that location. This may have interactions with datablock detection (**-d**).

--routines filename

-R filename
Causes a list of routines to be read from file **filename**, one per line as if they had been specified on the command line.

--addresses option

-a option
Determines if and what kind of address information should be dumped with the disassembly, if any. Note that this may make your output no longer intelligible to **xa(1).** The valid options are:

disabled
Dump source only with no address information. This

is the default.

enabled
Write the current address at the beginning of each line.

dump Write the current address at the beginning of each line, along with a hexdump of the bytes for which the statement was generated.

--colon-newline

-N

--no-colon-newline

-n A purely cosmetic option to determine how labels are emitted. Many people, including myself, prefer a listing where the label is given, then a tab, then the code (**-n**). Since this is *my* preference, it's the default. On the other hand, there are also many who prefer to have the label demarcated by a colon and a newline, and the code beginning indented on the next line. This is the way **d65** used to do it, and is still supported with **-N**.

--processor option

-p option
Specify the instruction set. Note that specifying an instruction set that permits and disassembles illegal and/or undocumented NMOS opcodes may make your output unintelligible to **xa(1)**. Only one may be specified. The valid options are:

standard-nmos6502
Only official opcodes will be recognized. This is the default.

r65c02
Opcodes specific to the Rockwell 65C02 (R65C02) will also be allowed.

all-nmos6502
Allows all 256 NMOS opcodes to be disassembled, whether documented or undocumented. Note that instructions generated by this mode are not guaranteed to work on all NMOS 6502s.

rational-nmos6502
Only allows "rational" undocumented instructions. This excludes ANE, SHA, SHS, SHY, SHX, LXA and LAXS. This is a judgment call.

useful-nmos6502
Only allows "useful" undocumented instructions. This excludes ANE, SHA, SHS, SHY, SHX, LXA, LAXS, NOOP and STP. This is a judgment call.

traditional-nmos6502
Only allows the most widely accepted undocumented

instructions based on combinations of ALU and RMW operations. This excludes ANE, SHA, SHS, SHY, SHX, LXA, LAXS, NOOP, STP, ARR, ASR, ANC, SBX and USBC. This is a judgment call.

--get-sa

-G

--no-get-sa xxxx

-g xxxx
Enables or disables automatic starting address detection. If enabled (the default), **dxa** looks at the first two bytes as a 16-bit word in 6502 little-endian format and considers that to be the starting address for the object, discarding them without further interpretation. This is very useful for Commodore computers in particular. If your binary does not have a starting address, you must specify one using **-g** or **--no-get-sa** followed by a hexadecimal address. The starting address will then be encoded into the output using *** =**.

--no-word-sa

-q

--word-sa

-Q Only relevant if automatic starting address detection is enabled. If so, the default is to also emit the starting address as a **.word** pseudo-op before the starting address indicated with *** =** so that it will be regenerated on re-assembly (**-Q**). Otherwise, if this option is disabled, the starting address word will not be re-emitted and will need to be tacked back on if the target requires it. If you specify an address with **-g**, then that address will be used here too.

--verbose

-v Enables verbose output, which may or may not be useful in the same way that Schroedinger's Cat may or may not be dead.

--help

-?

-V A quick summary of options.

The following options control how program code is scanned and determined to be a valid (or invalid) portion of a putative routine.

--datablock-detection option

-d option
This controls how the program automatically detects data blocks for addresses where no previous hints are

specified. Only one method may be specified. The valid options are:

poor As much as the object as possible will be listed as program code, even if there are illegal instructions present. This is the default.

strict
Assumes that all declared routines call and execute only valid instructions. If any portion of code declared as a routine leads to an address block containing illegal opcodes, a consistency error will occur and disassembly will stop.

skip-scanning
Program addresses that are not referenced by any routine will not be scanned for valid routines (thus data a priori).

--no-external-labels

-e

--external-labels

-E Controls whether labels should be generated for addresses outside of the program itself. The default is not to (i.e., leave the addresses absolute).

--address-tables option

-t option
Controls detection of address tables/dispatch tables. The following options are available:

ignore
Don't attempt to detect address tables.

detect-all
Address tables referencing any label will be detected.

detect-internal
Address tables with labels whose addresses lie within the program's address range will be detected. This is the default.

--no-suspect-jsr

-j

--suspect-jsr

-J These options indicate whether JSRs are always expected to return to the following instruction or not. This will affect how routines are parsed. For example, the Commodore 128 KERNAL has a routine called PRIMM that prints a null-terminated string directly following the JSR instruction, returning after the null byte. In this case, **-J** should be specified to alert the disassembler that this is possible. The default is to expect

"normal" JSRs (i.e., **-j**).

--no-one-byte-routines

-o

--one-byte-routines

-O These options permit or inhibit a single RTS, RTI or BRK instruction (or STP if enabled by the instruction set), or a conditional branch, from being automatically identified as a routine. The default is to inhibit this; specific cases may be selectively overridden with the **-r** option.

--no-stupid-jumps

-m

--stupid-jumps

-M These options consider jumps or branches to the current address (such as JMP *, BCC *) to be invalid or valid code depending on which is specified. Note that BVC * is always accepted as the V flag can sometimes be toggled by an external hardware signal. The default is to consider them invalid otherwise.

--no-allow-brk
-w

--allow-brk

-W These options control if BRK (or STP if enabled by the instruction set) should be treated as a valid exit from a routine, just like RTS or RTI. The default is not to do so.

--no-suspect-branches

-c

--suspect-branches

-C These options are rarely needed, but account for the case where a program may intentionally obfuscate its code using branches with unusual destination addresses like LDA #2:BEQ *-1. In the default case, this would be considered to be invalid and not treated as a routine (**-c**); if **-C** is specified, it would be accepted as valid.

BUGS/TO-DO

There are probably quite a few bugs yet to be found.

65816 opcodes are not (yet) supported.

The disassembler can easily be confused by the common idiom of tacking on BASIC text to call an appended ML routine. There probably should be a special case option for this. One workaround is to use the **--datablock** option and specify the range as unused (such as in the case of **10 SYS2061** (Commodore), giving **-b ?0801-080c** to ignore that range as data).

There are a few options Marko created that aren't hooked up to anything (and are not documented here on purpose). I might finish these later.

SEE ALSO

xa(1), file65(1), ldo65(1), printcbm(1), reloc65(1), uncpk(1)

AUTHOR

This manual page was written by Cameron Kaiser <ckaiser@floodgap.com>. **dxa** is based on **d65** 0.2.1 by Marko Mäkelä. Original package (C)1993, 1994, 2000 Marko Mäkelä. Additional changes (C)2006 Cameron Kaiser. **dxa** is maintained independently.

WEBSITE

http://www.floodgap.com/retrotech/xa/