

disassembler

COLLABORATORS

	TITLE : disassembler		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		August 26, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	disassembler	1
1.1	disassembler.doc	1
1.2	disassembler.library/--Background--	1
1.3	disassembler.library/Disassemble	1
1.4	disassembler.library/FindStartPosition	3

Chapter 1

disassembler

1.1 disassembler.doc

```
--Background--  
FindStartPosition()  
Disassemble()
```

1.2 disassembler.library/--Background--

PURPOSE

The purpose of this library is to provide an easy to use and powerful disassembler. It knows all MC68K processors starting with the 68000 up to the 68060, FPU and MMU instructions.

The code is MMU aware, i.e. won't try to access invalid or non-available memory, and makes use of the mmu.library if it is available. It will even work without the library, but it won't be able to validate its accesses and might, therefore, access invalid memory if the addresses passed in are incorrect.

The library functions are interrupt and supervisor callable, provided your functions - and the PutProc function passed in - is.

1.3 disassembler.library/Disassemble

NAME

Disassemble - disassemble MC68K object code to assembler

SYNOPSIS

```
nextpc = Disassemble ( disdata );  
d0      a0
```

```
APTR Disassemble ( struct DisData * );
```

FUNCTION

Disassembles the memory between the ds_From and ds_Upto fields writing the disassembled data out using the ds_PutProc function. The PC position is marked by a "*".

INPUTS

disdata - A structure specifying the details of the request.
It has to be filled out like the following:

```
struct DisData {  
    APTR    ds_From;  
    APTR    ds_UpTo;  
    APTR    ds_PC;  
    void    *ds_PutProc;  
    APTR    ds_UserData;  
    APTR    ds_UserBase;  
    UWORD   ds_Truncate;  
    UWORD   ds_reserved;  
};
```

ds_From is the memory location to start the disassembly from. This *MUST* be an even address.
ds_UpTo is the (exclusive) end of the memory region to disassemble.

ds_PC is a special address which is marked by an asterisk ("*") on disassembly. This feature can be used to mark the position of the PC.

void *PutProc is actually a function pointer. It is not declared as such to avoid unnecessary casting between compilers. Your code is called like this:

```
register d0 - the ASCII code to print. Line ends are marked with a  
             single line feed LF = 0x0a.  
register a1 - the contents of the ds_UserData field of the structure  
             above.  
register a3 - ditto.  
register a4 - the contents of ds_UserBase. This is most useful for  
             passing the pointer to the data space when using C or  
             other high level languages.
```

On exit, the routine may pass an updated a3 register back to the library which will be passed in a1 and a3 next time, similar to the exec RawDoFmt() function.

ds_UserData is the contents of registers a1 and a3 upon calling your routine. They are for your private use.

ds_UserBase is loaded to register a4 before calling the put procedure. Should be setup to be the _LinkerDB if the near data model is used.

ds_Truncate is the maximal line length to output. If this field is, for example, set to 80, longer lines will be truncated by the library to fit on the screen. If set to zero, no truncation takes place.

ds_reserved must be set to zero to support future extensions.

RESULTS

an updated pointer where the disassembly stopped. If you pass this pointer back in as "ds_From", the disassembler will automatically continue at the right place.

NOTES

this call will make use of the mmu.library if available. It won't access illegal memory or I/O space, provided the mmu table is setup correctly.

This routine is interrupt and supervisor callable, provided your provided bs_PutProc is. This makes this library function ideal for debugging tools and monitors.

BUGS

SEE ALSO

exec.library/RawDoFmt(), libraries/disassembler.h

1.4 disassembler.library/FindStartPosition

NAME

FindStartPosition - find an anchor point for disassembly

SYNOPSIS

```
start = FindStartPosition ( pc , min , max );
d0          a0    d0    d1
```

```
APTR FindStartPosition ( APTR , UWORD min , UWORD max );
```

FUNCTION

Given a PC address, this routine tries to find a useable address to start the disassembly from, in a way that the PC address is correctly included in the disassembly and that no, or as few illegal instructions as possible show up. The code in front of the PC is investigated, and a location between pc-min and pc-max is returned, with min<max. If no suitable address was found, the pc passed in is returned.

This makes this function an "go backwards" disassembly tool. This is usually problematic due to the CISC design of the MC68K, namely having instructions of various lengths.

INPUTS

pc - the anchor point. This is supposed to be known to be a valid address with a valid instruction. This instruction is guaranteed to show up in a disassembly started at the resulting address.

min - the minimum distance from the PC to start the search. Note that this must be positive, it will be subtracted from the "pc".

max - the maximal distance where the search will stop. This is again a positive number.

RESULTS

An address between pc-max and pc-min which, when used as start address for Disassemble(), will make the requested "pc" instruction part of the output stream.

NOTES

The purpose of this function is to find an anchor point for disassembling code "around" a given location. Use, as first step, the desired location to this procedure, then start disassembling at the result, using about max*2 bytes.

BUGS

SEE ALSO

Disassemble(), libraries/disassembler.h