APPENDIX A.

Subject : Dumpboj

Credits : By The Byte Works Inc.
Copyright 1986
Written by Phil Montoya

What Is Dumpobj?

Dumpobj is a powerful object module formatter. It's main task is to visually display object modules in easy to read format so that compiler writers can easily and quickly observe the output from the compiler. It also aids in debugging the linker because load modules can be displayed as well. In addition, -using several options in conjunction, Dumpobj can sucessfully display files of any type in hex format or 65816 disassembled format. This capability should not be confused with a disassembler. Dumpobj only displays information and is not designed to be used interactively.

Syntax

```
dumpobj [switches] filename [names=(segment_names,...)]
```

Options

+x Dump in hex format. The default is to dump the file in opcode format.

+d Dump in 65816 disassembley format. The default is to dump the file in opcode format.

-h Don't show headers. If the output format is hex, the header is now dumped in hex. For other output formats, the header is simply not printed. The default is to print headers.

-s Writes short headers instead of long headers. An abbreviated header is printed that is the same as a single segment output from the current scan utility. Used in conjunction with the -0 option the output will be equivilant as the current scan utility. The default is to print long headers.

-o The body of the segment is supressed. Instead, only headers are written. The default is to print the body.

-f Supresses check of file type. Normally, the utility only accepts object modules, load modules, and libraries. This allows any file type to be used. This can be handy for looking at files that have improper file types, but is really intended to let you dump files in hex format using this utility. The default is to check file types.

-a Normally, the left columns of the dumps include the displacement into the segment, the hex (for object format and assembley format) and the PC (for assembley format). This option suppresses that. The default is to ouput these values.

-m Default for 65816 disassembled listings is for full native mode. This causes the disassembly to start with short memory registers.

-i Default for 65816 disassembled listings is for full native mode. This causes the disassembly to start with short memory registers.

An optional **NAMES=(seg1 seg2...)** parameter allows dumping of selected segments from an object module.

Notes

1. You can either use '+' or '-' for each option. This maintains compatibility with the rest of the system.

   eg. **dumpboj -i +i -m +m <file>** is valid and is the same as **dumpobj <file>.**

2. It is best to use the **'-h'** option in conjunction with the **'-f'** option if the file is not in object module format. Anytime the header is printed, the file is assumed to be in object module format and information from the header is used to scan to the next segment. This may fail if the assumed header is not in correct form (ususally an error will occur). If the header is not printed and there is not

file checking, then a different mechanism is used to scan the file which will work sucessfully for any type of file.

3. The disassembler does it's best to correctly disassemble and object module. It can resync itself if it gets out of alignment in constant opcodes. It also tries to keep track of **rep** and **sep** instructions. Anytime an rep or sep with an immediate value occurs then the corresponding state of the registers is written via the **longa** and **longi** directives. These directives are also written at the begining of each segment, an reflect the state of the registers going into that segment. This will not work if an expression with a label was used as the operand of the rep or sep instruction.

   ie: (in original source)

   ```
   rep  #$30       works!
   sep  #$20*2     works!
   rep  #label     will not work!
   ```

4. For opcode dumps the expressions are written in **postfix** form. For disassembled dumps the expressions are written in **infix** form.

5. Our hope was that the disassembler option could produce, without modification, a source file that could be used by the assembler to create an equivalant object module. This is sort of the case. The differences lie in that the assembler resolves all relative branches and zero page equates so it is not possible to create local labels or use numonics on globally defined labels. There are simply some transformations that happen that don't allow us to create an exact duplicate of the source without making assumptions of how the object module was created. This is not what we want to do. On the other hand it is reasonable to assume that the two object modules once linked would produce identical load files.