Machine Language Support Programs

1.0 Introduction

ASM represents a significant investment in programming tools to enable the machine language programmer to analyze, modify and reconstruct programs on the Wang 2200 system.

Written in Basic, they utilize special atoms developed by the author, allowing us to view internally the structure of Wang.

Initially, the menu for the programs is called ASM, and can be loaded by 'LOAD RUN "ASM"'. The following screen will be displayed:

```
        Assembly Language Programs for Wang - Rev 2.0          ##############
                                                               ## COMPUTER ##
                                                               ## CONCEPTS ##
                        Press SF key to execute                ##############

 ' 1 - Editor       (Create eye readable machine code)
 ' 2 - Assembler    (Compiles source to object code)
 ' 3 - Adder        (Adds 4k of code to end of selected program)
 ' 4 - Patcher      (Merges object code into Basic or selected program)
 ' 5 - Searcher     (Searches selected file for Jump locations)
 ' 6 - Dissas       (Dis-Assembles Object code files)
 ' 7 - Symbols      (Enters symbology for use with Assembler)
 ' 8 - Debug        (Peek and Poke internal to Operating System)
 ' 9 - ATOMLIST     (Lists ATOMS and Sub-Verbs in ?? files)
 '10 - BREAKDM      (Breaks Down Data Memory to Array)
 '11 - CHDM         (Checksum Data Memory)
 '12 - Sequence     (Prints Bad sector Map for Winchesters)
 '13 - COMPDAT      (Compares Object files with memory)
 '14 - LDOBJECT     (Loads Binary files to CM and DM)
 '15 - CREF         (Cross Reference Generator for Object Files)

 '31 - START        (Return back to Master Menu)
```

Loading of any sub-program is done by simply pressing the Special Function key associated with that program. Of special note here is that the EDITOR and the ASSEMBLE program MUST be called vai the ASM program. ASM sets up some common variables defining the sizes of the partition we are running in.

2.0 ADDER

The ADDER program is a very simple program that just increases the size of a Control Memory load file by 4096 locations. This program is used to append a 4k block of space onto the end of a normal @@ program.

By doing this, we now have 4k more of Control Memory to write our programs in, or to enhance the storage capabilities of our programs.

When called, we simply type the name of the original program, followed by the disk address. Then we enter the new name for the program, followed by the target disk address.

Adder will procede to read the source file, transferring the data to the target file. When the entire source is transferred, ADDER appends blocks of code that are NOP control memory instructions. The trailer record is written, and the new source has an additional 4k of code. All that is left to do is to run this new file through PATCHER to append the correct checksums. The file may then be loaded normally.

```
******* Add to End 4k ********

Enter Original Program name : ? 324R
Enter Device Name  : 310

Starting record =    13408
Ending Record   =    13680
Number of Records =  273

Enter name for Output file : ? 33P
Enter device name       :  B10


Limits for 33P are:

Starting Sector Number =  1060
Ending sector #       =  1386
Number of Sectors     =  1
0000 000800 Record #  13408
0001 0000F0 Record #  13409
```

3.0 PATCHER

PATCHER is one of the most important of the utility programs, in that it allows us to merge an object file created by the Assembler into a current @@ or other machine language load file.

In addition, it lets us examine and change various locations in the file, as well as recalculating Data and Control Memory checksums.

When first loaded, the following display is seen:

```
          Patch Utility Menu          Revision 3.0

    '0 - Menu Listing
    '1 - Instructions for Patching
    '2 - Patch Disk
    '3 - Merge Files (Data Load DA formats)
    '4 - Load ASM menu

   '30 - Calculate Data Memory Checksums for file
   '31 - Calculate Control Memory Checksums for file
                                                    ###############
   *****  Use ONLY under guidance of CCC Field Service  *****   ## COMPUTER ##
                                                    ## CONCEPTS ##
 STOP     Press S.F. Key to begin                   ###############
```

SF key 0 will redisplay the Menu again, while SF key 4 will return us back to the main menu.

SF key 1 provides a crude refresh as to how to make patches.

3.1 SF key 2 - Patching the disk

Prior to us trying to merge an object file or recalculate checksums, we must load the paramaters of the file into PATCHER. Pressing SF 2 requests information as to what file, and to what disk are we going to do the modifications:

```
MOUNT (UNPROTECTED) DISK PLATTER TO BE PATCHED
  Address of Disk Platter (F10,R10, ...)  R10

  Name of File to be Modified: @@
------- PRE-SCANNING DISK FILE ---------
Loading: MVP (Multi-user) BASIC-2, Release 2.4 11/01/82- R18


                Load Records for @@

 1 Display Record

Data Memory     0000 to 0BFF
Control Memory 0010 to 5FFF
CM End Record  0000 to 000F


Location:       Data:
1C0010          576D4C
```

The above example shows a load of the file @@ from disk B10. Patcher will look up the file in the directory, and display any Header (TYPE) records. If no Revisions have been previously made, PATCHER appends the - R01 message to the type record. If previous revisions have been made, PATCHER increments the Rev level, and rewrites the first header. A scan is performed, accumulating all load locations, and when done, this is dumped to the CRT

3.1.1 Small patches

At the end of this first scan through, PATCHER will display the first Control Memory Address, along with the contents of that address on the CRT.

We may end this phase, by typing END followed by three spaces and a RETURN. If this is done, PATCHER will return to the Sub-menu, but all paramaters for Checksumming and Merging will be saved.

We can view or modify small areas of the target program manually by staying in display mode, and not typing END. Patcher will normally have displayed the first address as follows:

Location	Data
IC0010	576D4C

By pressing the RETURN key, the next sequential location will be brought up. If we wish to change the data, simply type in as many new digits as are required, when a RETURN key is depressed, the new data will be stored to disk.

If we wish to view a new Control Memory location, we simply type the Code letters 'IC', Instruction Counter, followed by the address we wish to view:

Location	Data
IC0010	IC4200
IC4200	87800F

Patcher may also view Data memory, if present in the source file. To do this, type 'PC', Program Counter, followed by the address we wish to view/modify.

Location	Data
IC0010	PC0010
PC0010	033312
PC0013	F46E52

Note that in the PC mode, the PC is incremented by three, while if in the IC mode, the IC is incremented by one.

Again, to exit from this, type END followed by three spaces and a RETURN

Since all changes are immediately done to the source file, be sure you really want to do the changes!

3.2 Merge Object files

The Assembler, when so instructed, will produce Object files from the Source files. These object files may then be merged with @@ or whatever, via this mode of operation.

You must have pre-scanned the target file by means of SF2, or an error message will result.

If the pre-scan had been performed, Mode 3 will request the Objects file disk address, as well as the name of the object file. After this is done, Mode 3 reads and displays the statuses of what it is merging. The following is a typical sequence:

```
Enter name of Patch file ? RED.B00
Enter name of disk : 318
 18 Load Records
Data Memory at 0000, 2 Locations
Data Memory at 018A, 2 Locations
Data Memory at 0192, 2 Locations
Data Memory at 01BE, 2 Locations
Data Memory at 0678, 6 Locations
Control Memory at 13EC, 5 Locations
Control Memory at 1734, 1 Location
Control Memory at 17FF, 1 Location
Control Memory at 1090, 80 Locations
```

3.3 Calculation of Data Memory Checksums - SF 30

Mode 30 is used to recalcultate the Data memory checksums of the target file. It first attempts to discern whether it is working with 2.4 or 2.3. If it cannot identify the Basic, it will abort the operations. If identification is established, the checksums are recalculated, and the file updated. Again, the pre-scan, SF2, must have been completed.

```
Calculate Data Memory Checksums
VERSION 2.4
Checksum at 08E0 is 94EE
Checksum at 08E2 is 9D5A

Modifying file - standby please

Data Memory Checksums are written

STOP 4015
```

3.4 Control Memory Checksums SF 31

To recalculate the correct Control Memory Checksums, this mode is required. When first invoked, it will print a small message, meant more as a caution note than a warning.

Pressing the CONTINUE key will result in the checksums to be recalculated and displayed.

After each checksum block has been calculated, the Old and New checksums for that block are displayed.

```
STOP SF'31 Calculate Control Memory Checksums - Key CONTINUE1000
:CONTINUE

  New            Old

5C6A8E917BB0  7EBE4113AFD0
7982A99C83B7  1A0634D635C4
9163D8557C42  0E78F64CBA80
7DEC8544081F  7DEC85440B1F
057E8AC5CA9D  1B3AF3DB866D
EEEB4643C518  EEEB4643C518
0000  16
End Program
Free Space= 46708
```

4.0 SYMBOLS

Another most useful program is the SYMBOLS program. SYMBOLS allows the user to assign symbology to control memory locations. This of course allows us to 'read' instructions with far more meaning than just bits.

The output of SYMBOLS is used by both the Dissassembler and the CREF programs.

When first invoked, the following display is present:

Symbolic entry for Disassembler Version 4.0

```
###############
## COMPUTER ##
## CONCEPTS ##
###############
```

Input File	Print Symbols	Listing Device
SYM.0000	Y	204
Disk	Sort by Address	
310	Y	

The author already filled in the blank areas with data, however, if a new SYMBOL file is to be created, then typing just a return key will get us right into the entry phase. However, most of the time we will be playing with a previously created file.

Enter the name of the symbol file. (I generally label them as SYM.---). Then tell the program where it lives. It will then ask if you wish to print the file, and if so, sorted and to what device the listing is to go to. Again, if we do not want to print, type N, followed by a RETURN, and we will load the file, then goto the entry phase.

If a file is printed, the format is shown on the next page.

Example of Print with Sorted Symbols

```
0001 HARD:RESET          0003 START:ADDRESS       000E SF'15RESETVECT      000F CKSMUPTOHERE        0081 MASTERCLRALL
0082 MASTERCLEARPART     008E RESET:VECTOR        0094 DOCHECKS:CMDM       00C1 PROMSF'15VECTOR     00C6 RESET:MSG
00E6 BACKGROUND3         0125 BEGIN:EXECUTION     0134 FNATOM              0138 VECTOR:2:ATOM       0139 ATOMRETURN
015F GETFIRSTCODE        01E2 BUMPSLMODE          01F7 HALT/STEP           021D HALT:STEP           0278 CLEARPRGCODE
027C SET09D6=PHPL        0284 REPORTERR2          0288 REPORTERR1          02E6 ERR:A02             02E9 PRINTERROR
02F9 FORMLETTER          0306 SETK=PL             0308 ERROR               0308 PROTECT             030F ERR:A06
0317 MATHERRORS          031C IMMEDIATE           031E ERR:A07             0339 ERR:A05             034B RSTENDOFPROG
034D CLEARRUNMODE        037E SET:LOAD/RUN        0381 RESOLVED            039E SCAN:SPACES         03A3 RANGE(0:9)
03A5 ASCII:(0:9)         03AF PHPLTOAR10          03B5 ASCII:(A:Z)         03CE SEARCH=COMMA        03CF SEARCH(R0)
03D5 SEARCHCR            03D7 SEARCH(             03D9 SEARCH)             03DB SEARCH=             03DD SEARCH'
03DF SEARCH(CE)          03E1 SEARCHP             03E3 SEARCH$             03E5 SEARCH"             03E7 SEARCHF
03E9 SEARCHR             03EB SEARCHS             03EB SEARCH:(R0)         03ED SEARCHT             03EF SEARCH/
03F1 SEARCH#             03F3 SEARCH:             03F5 SEARCH;             03F7 SEARCH<             03F9 SEARCH>
0400 CONVERTA(0:F)       0403 ASCIITOHEX          040F FIND(FD)            0410 SEARCH:REM:0D       042B SEARCH:R0
0436 COMPARE:AR01:10     044F WRITEATOMIN         0459 SHRINK:BUFFER       0461 2NDTRY:GETATOM      0462 MAKE:ATOM
046E NOT:ATOM            0471 SET(R0=FF)          0473 ATOMIZE             0476 GOTHRU:ATOMLIST     0488 SETR2R3=ARG
04C5 FIND:')'            04CD FIND:"="            04D1 TERMINATOR:0D       04D7 ERR:S13             04D9 SEARCH:END
04DF CK:TERMINATOR       04EA FINDATOM:(B2)TO     04ED BADLIST             04F3 DEVICET#            0504 XCHANGE
0514 GETHEXBYTE          051B ERR:S17             051D FROM:TO:ARGS        052C SEARCHLIST          057C FATOM(DB)%
```

4.1 Entry Phase

After the symbol table has been loaded, sorted and/or printed, we are ready to enter new symbols, or modify currently existing symbology. The following screen display will be present:

Symbol Entry Phase

Type END to dump buffer to disk

```
###############
## COMPUTER ##
## CONCEPTS ##
###############
```

Free Space 5752	Number of Symbols 806

Address	Symbol Tag
-	

For sake of discussion, let us say that we will assign location 4001 to be label FIND:K. Type the four digit address, followed by the label:

```
                Symbol Entry Phase                    ##############
                                                      ## COMPUTER ##
                Type END to dump buffer to disk       ## CONCEPTS ##
                                                      ##############

 Free Space  5743          Number of Symbols  807

 Address  |  Symbol Tag
----------|-----------------------------------------
  4001    |   FIND:K
----------|-----------------------------------------
```

After the label is entered, SYMBOLS searches the array, verifying that no duplicates exist, and will reclear the display. Now lets try to enter address 4001 again, the following display will result:

```
                Symbol Entry Phase                    ##############
                                                      ## COMPUTER ##
                Type END to dump buffer to disk       ## CONCEPTS ##
                                                      ##############

 Free Space  5733          Number of Symbols  808

 Address  |  Symbol Tag
----------|-----------------------------------------
  4001    |
----------|-----------------------------------------
 Duplicate found < 4001 FIND:K1 >

    Options
@ - Deletes Original Symbol
! - Change Address to last input

Change Address? (New Address or N or Options): ?
```

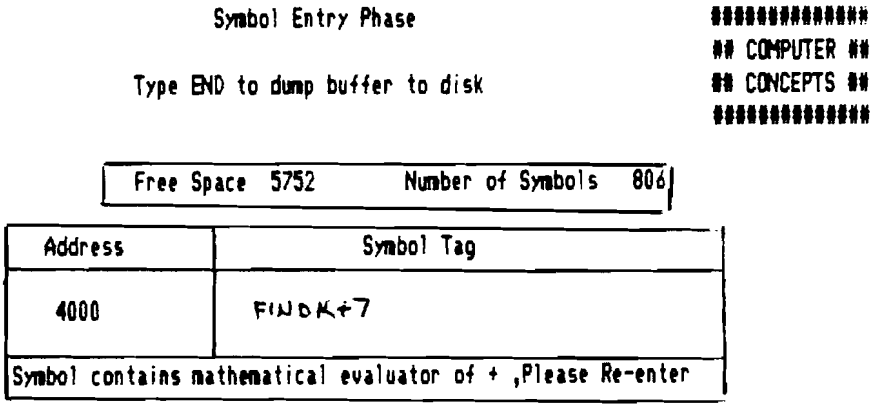
If we had erred, all we have to do is type return, and the Entry phase screen will be redisplayed. But sometimes, we do this on purpose. Lets say that I discovered that 4001 was not really FIND:K1. To delete a current symbol, I would then type 4001. SYMBOLS would report this immediately as a duplicate.

If I pressed the '@' symbol, this will delete 4001 entry.

Also, lets say that FIND:K1 was not really address 4001, but instead 4002. I would enter 4002 as the address, followed by the tag FIND:K1.

Again, symbols would report the duplicate, but now, I would type the symbol '!'. This changes the address tag from 4001, to the one I just input, 4002.

Symbols cannot contain any mathematical operator within the framework of the label. Though not important to the dissassembler, these mathametical operators would reek holy havoc in the Assembler. Therefore these are trapped, and a typical error display is shown below.



Also, the first character of a symbol cannot contain any numeric, or special code functions. The Assembler takes unkindly to that type of label. Therefore, we have another type of error display that checks these:

```
                Symbol Entry Phase                    ###############
                                                      ## COMPUTER ##
                Type END to dump buffer to disk       ## CONCEPTS ##
                                                      ###############

   Free Space  5752          Number of Symbols  806

| Address |  Symbol Tag  |
|---------|--------------|
|  4000   |  2FINDK      |

Symbol cannot start with $'01234567890%. Data, Re-enter please
```

4.2 *Dumping the Symbol table*

When completely done editing the current symbol table, type END to the prompt for new address. The SYMBOLS program will display the closing screen, and request information about how to dump the data. In almost all cases, I recommend that you allow the full array size to be dumped. This normally gives us room for about 1500 symbols.

```
                Dump Symbol Buffer to Disk            ###############
                                                      ## COMPUTER ##
                                                      ## CONCEPTS ##
                                                      ###############

| File Name | Disk Address | Full Array Size |
|-----------|--------------|-----------------|
| SYM.2BBL  |     B10      |        Y        |

          Records Required =  42
   Current file SYM.2BBL has 66 records
```

5.0 CREF Cross Reference File Program

The Cross Reference File programs is yet another tool to allow us to break apart the code of the Wang 2200 system. These two programs, CREF and CREF:LPI, have seperate functions.

The initial CREF:MENU is displayed:

```
                                                  ##############
                                                  ## COMPUTER ##
                    Cross Reference Menu          ## CONCEPTS ##
                                                  ##############

  SF'0  Cross Reference LPI Instructions
  SF'1  Cross Reference JMP,JSR and Branch Instructions
  SF'2  Return to ASM menu
```

5.1 CREF

The main CREF program allows us to accumulate all Jump, JSR and Branch instruction target addresses, and display these on the output devices.

By defining what paramaters we wish, we can display all subroutines, and how many times they are called, and by whom. This eases the burden of finding out if areas are used by other rouines, and greatly simplifies our attempt at cracking code.

We also may enter a Symbol table into CREF. When the dump is proceeding, the addresses, and our symbology will be printed.

CREF uses the area outside of the catalo on disk to store the matrix table. As many accesses to the disk are normally required, we recommend that the disk not be a floppy.

The following page shows a typical screen for the CREF program:

File Name	Start Address	End Address	Min Accesses
33 B10	0000	4FFF	1
Symbol File	Jump Enable	JSR Enable	Branch Enable
SYM.824 310	Y	Y	Y
)= Address	(= Address	CREF file name	Listing
0000	4FFF	(None) 310	204

Zeroing Reference Area 14234 to 14555

The start address defined in the above screen references the target file. In the above example, we want to start at location 0000. However, we could set this to any valid address within the file. Compilation of the targets would start at that location.

The end address defines the ending control memory location. The search for JMP, JSR and Branch instructions will cease at that address.

Minimum accesses simply means that the display, at the end of the scan, will only display those locations that have been accessed by that many or more times. When breaking bran new code, I also set this number to about 6. The list of numbers produced will allow me to quickly make some intelligence about the program.

The symbol file entry is optional. However, entering a symbol file makes for a clearer CREF listing.

The next three questions, Jump Enable, JSR Enable and Branch Enable, allows us to customize what type of CREFwe are looking for. If we were just interested in sub-routines, then we would only answer 'Y' to the JSR Enable. Only JSR instructions would be accumulated.

The next two questions defines the limits of our targets. We will enter a specific number that means if the JSR, JMP or BRANCH instruction, vectors us to a address less than address 1, or greater than address 2, not to include these in the accumulation.

We also have the capability to Save the accumulation file. Normally I do not. Finally, answer where you want the dump.

Again, most of the time we print to the 204 device. CREF zeroes out the reference matrix, and starts the accumulation progress. After a long period of time, the printout should appear. Below is a sample of the CREF output.

```
Cross Reference CM 0000 to 4FFF File 22  Accesses > 0              Page 1
------------------------------------------------------------------------------

00C1                    00B3 00BE
00D1  RESET:MSG         00C0
00E1                    00DF
00E6  RESETCO:OUT       0081 0098 00CE
00ED  BACKGROUND1       01EC 0209 021F 18F8 1ADC
00EF  BACKGROUND2       0095 00D0 0146 0169 02DF 02E0 02E1 0A62 1A26 1C0B 1C16 1C1C 26FF 4892
00F1  BACKGROUND3       0085 0210 0292 02E2 02F6 1A23 27EC
00F8                    00F6
00FE                    00FB
00FF                    00FD
0110                    010E
0117                    0115
0124                    0118
0128                    0125 0126
012D                    012A 0128
0130  BEGIN:EXECUTION   27CA
013F  FNATOM            014F 01F2 023C 0241 0243 0244 0246 0249 031E 1867 1C5F 1D95 3EA9 4717
0143  VECTOR:2:ATOM     0141 0149
0144  ATOMRETURN        015E 0160 0161 03A0 03A2 03A4 03A8
0145                    0137
014E                    023D
0150                    013D
0151                    277D
015F                    015D
0167                    0152
```

5.2 CREF:LPI

The CREF:LPI program is similar to the CREF program, except only LPI instructions are referenced.

This aids us in determining Data Memory location references, and in assisting us in determining Control Memory List functions.

Since the output is identical to CREF, only the CREF:LPI screen is shown below.

```
                                                  ###############
                                                  ## COMPUTER ##
Cross Reference Object Files - LPI Instruction    ## CONCEPTS ##
                                                  ###############
```

File Name	Start Address	End Address	Min Accesses
22 B10	0000	4FFF	1
Symbol File	LPI Instructions are Enabled		
SYM.224 310			
>= Address	<= Address	CREF file name	Listing
0000	4FFF	(None) 310	204

Zeroing Reference Area 14234 to 14555