NAME

assembler, assem, assem16to8, assem24to8, assem32to8 - translate symbolic microcode into hex integer file(s)

SYNOPSIS

assembler < xxx.as > xxx.dat
assem xxx
assem16to8 xxx
assem24to8 xxx
assem32to8 xxx

DESCRIPTION

This program translates symbolic microcode instructions into a file of hex integers. An optional listing file can also be produced which reproduces the source file with the addition of the address and value for each microinstruction. The *assembler* output must then be processed by the *dat2ntl* program to format this file into one that can be sent to the PROM programmer. The *dat2ntl* program is described in a separate man-page. The *assembler* accommodates a large number of microinstruction formats. The user can (must) define individual keywords and instruction formats in a specification file.

There are several shell scripts which run both *assembler* and *dat2ntl* so these programs do not have to be run separately. These scripts pipe the output of *assembler* through the program *dat2ntl*.

By the recommended convention, the source file name has an extension of .as, i.e., it is of the following form:

xxx.as

The specification file name is of the form:

xxx.sp

The listing file name should be of the form:

xxx.list

USE OF THE PROGRAM

To use the program one must create a specification file and a source file.

The specification file, xxx.sp, contains the declaration of the command names and the bits which each command asserts. It also specifies which bits will be used for the address field, the instruction size, and, optionally, whether the default assertion for signals is low or high. The microinstructions must all be of the same length and can only reference a single address.

The source file, xxx.as, contains the assembly code which uses the declarations given in xxx.sp. It must include a #SPEC_FILE statement in order to know where to look for these declarations. It may also include a #LIST_FILE statement which provides a file name for the *assembler* listing, a #SET_ADDRESS statement to tell the *assembler* the beginning address for the assembled code, and a #NEW_PROGRAM statement to tell the assembler to treat the following text as a completely new program. Any other command statement - "#command_name = value ;" - will be passed through the *assembler*.

There can be multiple #SET_ADDRESS statements interspersed within the code so that the address can be changed anywhere in the code. This is useful for **subroutines** or when a certain part of the code needs to be put in a specific position in memory. The formats of these statements can be obtained from the examples or from the man page for *dat2ntl*.

Note that the LIST_FILE produced by the *assembler* can be used in your project reports.

WHAT THE ASSEMBLER DOES

The *assembler* operates in two passes. It first opens xxx.as. From this it determines the name of your specification file, xxx.sp, and processes this file to determine your instruction formats. The *assembler* then proceeds to process xxx.as. During this first pass, the values of all address labels are evaluated and stored. Except for forward address references, the values of the microinstructions are also determined during this first pass. During the second pass, the values of the forward referenced address labels are included in the final microinstruction values; and the final output is produced along with the optional listing file.

To produce an output number for a microinstruction, the *assembler* first evaluates a token. It then shifts the value left so that it lines up with the specified field and logically ORs the value into the output number. While doing this it checks to see if the value is wholly contained within the specified field. If not, it produces a warning error comment. It also produces a (different) warning if successive tokens cause the same bit in the output number to be specified more than once. When the statement terminating semicolon is reached, the *assembler* increments the address counter and processes the next microinstruction.

You may include multiple, separate programs within the same source file. All of these programs must use the same specification file. These programs are delineated by the inclusion of a command statement,

        # new_program = prom_address

This command statement will be translated into a

        # load_address = prom_address

statement.

HOW TO RUN THE PROGRAM

The *assembler* by itself translates its input file into a data file of hex integers. This file then has to be processed by *dat2ntl* to create the file, xxx.ntl, before you can program a PROM on a DATA I/O programmer.

            assembler <xxx.as >xxx.dat

There are several shell scripts which run both *assembler* and *dat2ntl* so these programs do not have to be run separately. These scripts pipe the output of *assembler* through the program *dat2ntl.*

The following shell scripts all process xxx.as:

assem xxx   Creates xxx.ntl

assem16to8 xxx   Creates byt0xxx.ntl and byt1xxx.ntl

assem24to8 xxx   Creates byt0xxx.ntl through byt2xxx.ntl

assem32to8 xxx   Creates byt0xxx.ntl through byt3xxx.ntl

INPUT FILE FORMAT

The *assembler* parses its input into tokens. White space, i.e., spaces, tabs, returns, and newlines, are ignored, although your file must end with a newline or else the last line will not be read.

The parser is case insensitive. Internally, all upper case characters are mapped into lower case characters. Since the names of the specification file, xxx.sp, and of the listing file, xxx.list, are specified in the source

file, xxx.as, the actual UNIX file names must not include upper case characters.

Legal characters for labels or symbols include a through z, 0 through 9, . (period), _ (underscore), / (slash), and %. Labels and symbols must include at least one nonnumeric character.

The same label cannot be used as an address label and a command symbol, as the *assembler* will consider this to be a redeclaration error.

Comments are delimited as in C.
For example, /* this is a comment */
/*
 * This is also
 * a comment.
 */

Integers are normally interpreted as decimal numbers. If you precede an integer with %b or %h, then that integer will be interpreted as a binary or hex number respectively. For example, %b10001 will be read as 17 and %h1c will be read as 28.

CAUTION - The integer in a #SET_ADDRESS, #LOAD_ADDRESS, or #NEW_PROGRAM command statement is ALWAYS interpreted as a HEX number.

All statements must be terminated with a ; (semicolon). The *assembler* attempts to give meaningful error comments. However, if you do not understand the error comment, check for a missing semicolon.

SPECIFICATION FILE

Fields are specified by the construct <n:m> or <n> where n > m and both n and m are in the range 0 through 31. This construct is used to simplify the specification of symbol values and to enable range tests.

The first entry in the xxx.sp should specify the instruction length. The maximum allowed length for an instruction is 32 bits, and this is also the default.

        op <n:0>;

You must specify the address field. You can only have a single address, and the field must be a single contiguous block of bits. If you must have the address field in split fields, then you can simply wire your PROM outputs accordingly.

        address op <n:m>;

You may optionally specify a field into which integer values are placed by including

        value op <n:m>;

The default value for a field is all zeros, and ones are ORed into the field. If you wish, you may specify the default value to be all ones by

        assert_low op <n:m>;

Then zeros will be ORed into those fields. Note that you must also invert the symbol definitions to be used in that field. This has an effect similar to inverting those PROM outputs.

You may specify noncontiguous bit fields, e.g.,

        assert_low op <7:6> op <3:2> op <0>;

Command symbols are defined by

        NAME op <n:m> = NUMBER;   or
        NAME op <n> = NUMBER;

where NAME is any symbol you wish and NUMBER is an integer. Non-contiguous bit fields can be speci-
fied, e.g.,

        NAME op <7:6> = 2 op <4:3> = 1;

One can also define new NAMES in terms of previously defined NAMES. Or one can mix NAMES with
bit field specifications. Forward references are not allowed. Examples are:

        load   op<3> = 1;
        shift   op<2:0> = %b110;
        new   load shift;
        new1   op<2:0> = 2 load;
        new2   op<2:0> = 2 op<3> = 1;

As a special case, you may wish to define a symbol (or symbols) that does not do anything, i.e., contribute
to the output value or produce an error comment. You accomplish this by

        NAME nop;

Look at the example file to see how this feature can be used to make your microcode more readable.

SOURCE FILE

        The source file, xxx.as, consists of command statements, microcode statements, and, of course, comments.
        Command statements begin with a # and are of the form

        # key_word = value;

The first statement in your source file must be

        # SPEC_FILE = xxx.sp;

where xxx can be anything you like. Note that the actual UNIX file name must not include upper case char-
acters as the *assembler* maps all upper case characters to lower case characters.

Normally a listing file is desired, so you should include the listing file name specification.

        # LIST_FILE = xxx.list;

You may include the following statements to specify the beginning address for your program. If you omit
these two statements, then the *assembler* assumes the default of zero for both NUMBER_SA and NUM-
BER_NP.

        # SET_ADDRESS = NUMBER_SA;

        # NEW_PROGRAM = NUMBER_NP;

This statement is transformed into a

LOAD_ADDRESS = NUMBER_SA + NUMBER_NP;

statement which is passed through so the *dat2ntl* program can use it to determine where your program is to be loaded into your PROM.

If you wish to have your program loaded into your PROM at a different address, you can also include

LOAD_ADDRESS = NUMBER;

to specify that. The effects of #SET_ADDRESS and #LOAD_ADDRESS differ in that #SET_ADDRESS is interpreted by the *assembler* to set the internal location counter. All label references are thus affected. The #LOAD_ADDRESS statement, on the other hand, is not used by the *assembler* at all, but merely affects the PROM locations that are programmed.

The #NEW_PROGRAM statement is used to delineate multiple programs within the same file and to specify the beginning PROM address for this new program. Thus, it is possible to have multiple programs in a single PROM and select a given program by switching the high order address bits.

Other command statements to be interpreted by the *dat2ntl* program may be included and will simply be passed through to the output without affecting your assembly.

You are required to include a #SPEC_FILE statement before any assembly code, but you are not required to include a #LIST_FILE statement or a #SET_ADDRESS statement. If no #LIST_FILE statement is included, no list file will be created. If no #SET_ADDRESS statement appears before the first line of the microcode, then the *assembler* will start at address zero. The *dat2ntl* program requires either a #SET_ADDRESS or #LOAD_ADDRESS statement. You should include one or the other right after the #SPEC_FILE statement.

Microcode instructions are of the form

addr_label : symbol1 symbol2 address;

A token is interpreted as defining an address label if it is followed by a : (colon). You need not have an address label defined on every instruction. Your microcode instruction can consist of an arbitrary number of symbols, but you may have only one address label among them. The *assembler* will evaluate each token and OR it into the field as specified in xxx.sp.

Integers can be put directly into the xxx.as file if a value field has been specified. The integers must fit within the value field. As in the xxx.sp file, they will be read in decimal unless preceded by a %h or %b.

By using the assert_low command in the xxx.sp file, all the specified bits will default to ones instead of zeros.

Remember that all statements must be terminated by a ; (semicolon).

FILES

/mit/6.111/handouts/labs/lab3.s95/mcu.as	MCU test program
/mit/6.111/handouts/labs/lab3.s95/mcu.sp	MCU spec. file
/mit/6.111/handouts/labs/lab3.s95/mcu.pal	MCU test PAL
/mit/6.111/prom/examples/encr.as	
/mit/6.111/prom/examples/encr.sp	
/mit/6.111/prom/examples/encr.list	

SEE ALSO
      prom(1), dat2ntl(1)

BUGS