HILEVEL Technology, Inc.

Q'n Apps #E32

Q: What is the Symbolic Addressing Mode?

Symbolic Addressing in Program Mode

The vector comment field can be employed for symbolic addressing. Certain rules are enforced, primarily to extend program mode to QuickLoad in AutoTest. This beta feature is invoked by the SpScript keyword óAllowSymbolicAddressInCommentsô. The following Rules & Regulations apply:

A. The first comment line (vector 0 comment) must start with the keyword ó_PrgCmdô, indicating that comments have program commands.

B. First line of program must be PAGE 0, and second line of program must be JMP 2. Note: these program commands are automatically generated (the aforementioned keyword does the invocation).

C. Vector 2 comment must begin with ó:BEGINô (which is a ólabelô preceded by the key delimiter ó:ô). Again, the ETS2k generates it automatically.

D. Last vector comment indicates last vector by the keyword ó_LastVecôï inserted automatically. This, however, is not a label; just a comment convenient when the feature of comment search is used. Similarly, the StopVector will have the comment ó_StopVectorô inserted. (CommentSearch makes these comments an operating convenience).

E. The comment for the last vector minus 1 must begin with ':END'. This label is also generated automatically. Between óBEGINô and óENDô (inclusive) is the program space accessible as references by the various branch instructions of the program.

By these five simple rules, of which conformation and/or generation is automatic, lies the foundation for applying program mode to QuickLoad in AutoTest. In addition, symbolic addressing must be adhered to throughout. While we can run vector programs where symbolic programming is only adhered to partially, such programs cannot be applied to AutoTest if QuickLoad is desired.

The syntax for the program itself is simple and summarized by the following rules:

A. Only branch instructions associated with labels (JMP, CJMP, CALL, CCALL) need to be considered for programming in the comment field. The LOOP instruction has by our new convention no label associated with it. It is called LOOP@, indicating looping to itself. If the user desires the old convention, we have provided equivalency by invoking the keyword óUseOldProgramRulesô.

B. To assign a symbolic label to a branch instruction, just type a ó$ô followed by ó=ô, ó+ô, ó-ó, or LabelName. The ó=ô signifies branch to itself (e.g. waiting for a condition), while ó+ô and ó-ó signifies relative addressing - that is, relative to the location of the branch instruction (the +/- will be followed by an integer which determines the relative branch).

LabelName is any label of any size, of which the first 15 characters have significance (i.e. 16th character and characters beyond are ignored).

C. The LabelName associated with a vector must appear in the comment field at that vector and be preceded by the delimiter ‘;’. The label name itself could be any alphanumeric string of which the first fifteen characters have significance.

The vector display to the right illustrates the concept.

Note: the line with ‘;_StopVec’, indicating the Stop Vector, which is not the same as Last Vector. ETS2k warns the user of this deviation from the rules, but allows it to run anyway. The warning message will only appear one time during the entire session.

This comment file should be saved as a .PRA file when uploading vectors.

Before pressing Run:

Vector Address (Hex)	Program	Split Cycle	Comments
00000	NOOP	00000000 00000000	_PrgCmd
00001	NOOP	00000001 00000001	
00002	LOADX 00	00000002 00000002	
00003	LOAD 0FFF	00000003 00000003	
00004	NOOP	00000004 00000004	
00005	NOOP	00000005 00000005	just a comment
00006	NOOP	00000006 00000006	
00007	CJMP 0000	00000007 00000007	$Alabel more comment
00008	JMP 0000	00000008 00000008	$+3;jump to adr +3
00009	NOOP	00000008 00000008	
0000A	NOOP	00000008 00000008	:Alabel
0000B	NOOP	00000008 00000008	
0000C	LOOP@	00000008 00000008	
0000D	NOOP	00000008 00000008	
0000E	NOOP	00000008 00000008	
0000F	NOOP	00000008 00000008	
00010	NOOP	00000008 00000008	

After pressing the Run button, the display changes to:

Vector Address (Hex)	Program	Split Cycle	Comments
00000	PAGE 00	00000000 00000000	_PrgCmd
00001	JMP 0002	00000001 00000001	
00002	LOADX 00	00000002 00000002	:BEGIN
00003	LOAD 0FFF	00000003 00000003	
00004	NOOP	00000004 00000004	
00005	NOOP	00000005 00000005	just a comment
00006	NOOP	00000006 00000006	
00007	CJMP 000A	00000007 00000007	$Alabel more comment
00008	JMP 000B	00000008 00000008	$+3;jump to adr +3
00009	NOOP	00000008 00000008	
0000A	NOOP	00000008 00000008	:Alabel
0000B	NOOP	00000008 00000008	
0000C	LOOP@	00000008 00000008	
0000D	NOOP	00000008 00000008	
0000E	NOOP	00000008 00000008	_StopVec
0000F	NOOP	00000008 00000008	:END
00010	NOOP	00000008 00000008	_LastVec

Importantly, this real time compilation (perhaps more correctly *assembly*) slows down the running process as a function of several factors, primarily the number of lines of comments and the total number of comments. While, the translation will only occur as required (a subsequent pressing of the RUN button will skip the compilation process), it slows down the run when a comment is added or changed. We understand the essence of the element of time during the debugging process, and have done our utmost to mitigate this timing predicament. However, assembly is never done instantly and the price to pay may warrant temporary deviations from the norm. Do this by changing the aforementioned keyword ‘;_PrgCmd’ to ‘;PrgCmd’, and the software will behave as if no compilation existed. Then, when the job nears completion, you can reintroduce ‘;_PrgCmd’ and finish off the job with a program well documented and easily enhanced and modified.

Also See:

Q'nApp #E7: Vector Looping
Q'nApp #E8: Pattern Matching
Q'nApp #E15: Special Script
Q'nApp #E33: MASK and UNMSK
Q'nApp #E39: PRG file attachment
Q'nApp #E44: Vector files w/o path