NOTE: This Technical Note has been retired. Please see the Technical Notes page for current documentation.

Technical Note PT545

MPW Assembler Q&As

CONTENTS

This Technical Note contains a collection of archived Q&As relating to a specific topic--questions sent the Developer Support Center (DSC) along with answers from the DSC engineers. Current Q&A's can be found on the Macintosh Technical Q&A's web site.

[Oct 01 1990]

Using "far" addressing model with MPW Assembler

Date Written: 6/7/91

Last reviewed: 8/1/92

Now that MPW includes the "far" addressing model, how does one use the assembler with the far model? For example, how does one generate a "jsr.l symbol" that the linker will recognize as a "far" reference?

The MPW Assembler does support the new "far" runtime model; it's documented in the "MPW 3.2 Run-Time Architecture" release note, included with recent versions of MPW that support the new model. (The path to this note on E.T.O. #4 is E.T.O. #4:Tools - Objects:Macintosh Programmer's Workshop:MPW QR4:QR4 Notes:MPW QR4 Run-Time Architecture.)

The specific example to your parenthetical question is "JSR (symbol).L", but you should see the release note for background information and important caveats.

MPW 3.2 ASM generic instruction conversions

Date Written: 8/21/91

Last reviewed: 6/14/93

Where are the MPW 3.2 Assembler "generic instruction conversions"? After upgrading from MPW Assembler 3.1 to 3.2, I noticed while debugging some of my code that an Adda instruction was converted to Lea for me. While I appreciate this (it saves cycles) I didn't expect it nor can I find where it's documented. The cover letter that came with 3.2 says that Asm has "new optimizations" but doesn't elaborate. I have found Appendix A in the 3.0 documentation that describes some obvious optimizations. Where is the corresponding list of optimizations for 3.2?

You correctly noted that the 3.2 Assembler supports optimization of instructions which were previously not optimized. The following additional optimizations are available:

```
ADD   #<Data>,An   [-8 <= data <= -1]  =====>   SUBQ   #-<Data>,An
ADDA  #<Data>,An   [-8 <= data <= -1]  =====>   SUBQ   #-<Data>,An
ADDI  #<Data>,An   [-8 <= data <= -1]  =====>   SUBQ   #-<Data>,An

SUB   #<Data>,An   [-8 <= data <= -1]  =====>   ADDQ   #-<Data>,An
SUBA  #<Data>,An   [-8 <= data <= -1]  =====>   ADDQ   #-<Data>,An
```

These optimizations only occur when the Assembler understands the operand to be a negative number. Since the Assembler operates on 32-bit arithmetic, the operand value must be written as a negative number to activate the optimization, regardless of the size (B, W, L) of the instruction. Thus, in a byte operation, the operand must specify "-1" instead of "$FF," even though, unoptimized, they would both compile to the same instruction.

```
ADDI  #<Data>,<EA3>   [+1 <= data <= +8]  =====>   ADDQ   #<Data>,<EA3>
SUBI   #<Data>,<EA3>   [+1 <= data <= +8]  =====>   SUBQ   #<Data>,<EA3>
```

MPW Assembler and "~" character

Date Written: 10/17/91

Last reviewed: 6/14/93

When I try to turn on Macintosh interrupts after simple diagnostics, the MPW Assembler reports an error, as follows:

```
# 303:     ANDI.W  #~$0700,SR
### Error  54 ### Illegal/missing operands or opcode not allowed for target
```

Is there a way to get this to work?

Using the tilde character on a one's complement operation in MPW Assembler can give confusing results. The assembler,

in its infinite wisdom, treats all constants as signed 32-bit values. Therefore, $0700 is really treated as $00000700. The one's complement of this is $FFFFF8FF, which is too large a value for an ANDI.W instruction. The syntax that will work is:

Back to top

Downloadables

Acrobat version of this Note (K). Download
