

z390 Macro Pseudo Code as v1.2.00

The z390 macro processor (mz390.java) now has support for macro pseudo code which is generated in a cache memory buffer during conditional macro code source statement parsing for AGO, AIF, SETA, SETB, and SETC statements. If the cache size is exceeded, macro pseudo code is replaced on a least recently used basis resulting in the pseudo code being regenerated if required. When a macro statement with existing pseudo code is executed again, no source parsing is performed and instead the pseudo code is executed.

Initial results on large macro assembly with over 180 macros executing over a million lines of macro code being executed, indicates a 3 to 1 improvement in speed reducing 45 second process to 14 seconds. The speed improvement is achieved by eliminating the overhead of source statement parsing and symbol lookup.

Several new options have been added to z390 as part of Macro Pseudo Code support:

1. **PC** – default option to generate macro pseudo code
2. **NOPC** – turns off macro pseudo code and just used source parsing
3. **OPTPC** – optimize the generated macro pseudo code as follows:
 - a. Replace **PUSHA** arithmetic constants and **PUSHC** string constants with immediate constant in pseudo operations such as **ADD**, **SUB**.
 - b. Replace **PUSHV &var**, **ADD 1**, **STORV &var** with **INC &var**.
 - c. Replace **PUSHV &var**, **SUB 1**, **STORV &var** with **DEC &var**.
4. **MAXPC(40000)** – default cache for macro pseudo code operations. For maximum benefit this size must be large enough to hold macro working set of pseudo code that is executed many times.
5. **TRACEP** – generate detail macro pseudo code trace of generated and executed pseudo code operations including the value of input and output variables for each pseudo operation. This trace is directed to new TRM trace file and includes the **TRACEM** option showing all conditional macro statements executed and all generated **BAL** statements with both macro source line references and generated statement number references.

The entire macro pseudo code support consists of about 2,000 lines of Java code added to the mz390.java source. The macro pseudo operation codes are single operand instructions which perform operations on the same push/pop stack used by the source expression parser. Both the source statement parser and the pseudo code Share execution routines. The operation codes are defined as follows in mz390.java:

```
byte pc_op_ago = 1; // branch based on stack index value
byte pc_op_aif = 2; // branch if stack value not 0
byte pc_op_pushv = 3; // push var on stack
byte pc_op_pushvs = 4; // push var(sub) on stack
byte pc_op_pusha = 5; // push seta self defining term
```

```

byte pc_op_pushc = 6; // push setc string constant
byte pc_op_concat = 7; // concatenate setc constant
byte pc_op_storv = 8; // store scalar set var
byte pc_op_storvs = 9; // store subscripted set var
byte pc_op_storvn = 10; // store next multiple value in var
byte pc_op_add = 11; // add
byte pc_op_sub = 12; // subtract
byte pc_op_mpy = 13; // multiply
byte pc_op_div = 14; // divide
byte pc_op_compeq = 15; // compare equal
byte pc_op_compge = 16; // compare greater than or equal
byte pc_op_compgt = 17; // compare greater than
byte pc_op_comple = 18; // compare greater less than or equal
byte pc_op_complt = 19; // compare equal
byte pc_op_compne = 20; // compare greater than or equal
byte pc_op_ucomp = 21; // unary compliment value on stack
byte pc_op_dup = 22; // duplicate string
byte pc_op_sublst = 23; // calculate setc sublist
byte pc_op_substr = 24; // calculate setc substring
byte pc_op_inc = 25; // inc var/varsub
byte pc_op_dec = 26; // dec var/varsub
byte pc_op_pushd = 27; // push scalar dynamic var
byte pc_op_pushds = 28; // push subscripted dynamic var
byte pc_op_stord = 29; // store scalar dynamic var
byte pc_op_stords = 30; // store subscripted dynamic var
byte pc_op_pfx_a = 31; // A' lookahead defined symbol
byte pc_op_pfx_d = 32; // D' ordinary defined symbol
byte pc_op_pfx_i = 33; // I' integer count
byte pc_op_pfx_k = 34; // K' character count
byte pc_op_pfx_l = 35; // L' ordinary symbol length
byte pc_op_pfx_n = 36; // N' number of sublist operands
byte pc_op_pfx_o = 37; // O' operator
byte pc_op_pfx_s = 38; // S' scale factor
byte pc_op_pfx_t = 39; // T' symbol type
byte pc_op_pushs = 40; // push symbol value abs value if found else 0
byte pc_op_a2b = 45; // convert value to binary string (3 = '11')
byte pc_op_a2c = 46; // convert value to character string (240 = '1')
byte pc_op_a2d = 47; // convert value to decimal string (1 = '1')
byte pc_op_a2x = 48; // convert value to hex string (240 = 'F0')
byte pc_op_and = 49; // logical and (NC)
byte pc_op_b2a = 50; // convert binary string to value (B2A('100') = 4)
byte pc_op_b2c = 51; // convert binary string to character string
byte pc_op_b2d = 52; // convert binary string to decimal string
byte pc_op_b2x = 53; // convert binary string to hex string
byte pc_op_c2a = 54; // convert characters to value
byte pc_op_c2b = 55; // convert character string to binary string

```

```

byte pc_op_c2d = 56; // convert character string to decimal string
byte pc_op_c2x = 57; // convert character string to hex string
byte pc_op_d2a = 58; // convert decimal string to value
byte pc_op_d2b = 59; // convert decimal string to binary string
byte pc_op_d2c = 60; // convert decimal string to character string
byte pc_op_d2x = 61; // convert decimal string to hex string
byte pc_op_dclen = 62; // length of string after reducing double ',&
byte pc_op_dcval = 63; // return string with double ' and & reduced
byte pc_op_dcequo = 64; // return string without first and last '
byte pc_op_double = 65; // double quotes and & in string (NC)
byte pc_op_find = 66; // return index of any char in string
byte pc_op_index = 67; // return index of string2 found in string1
byte pc_op_isbin = 68; // return 1 if valid binary string else 0
byte pc_op_isdec = 69; // return 1 if valid decimal string else 0
byte pc_op_ishex = 70; // return 1 if valid hex string else 0
byte pc_op_issym = 71; // return 1 if valid character string for sym
byte pc_op_lower = 72; // return lower case string (NC)
byte pc_op_not = 73; // logical or arithmetic not (NC)
byte pc_op_or = 74; // logical or (NC)
byte pc_op_upper = 75; // return upper case string (NC)
byte pc_op_signed = 76; // return decimal string with minus if negative
byte pc_op_sla = 77; // shift left arithmetic (2 SLA 1 = 4)
byte pc_op_sll = 78; // shift left logical (2 SLL 1 = 4)
byte pc_op_sra = 79; // shift right arithmetic (4 SRA 1 = 2)
byte pc_op_srl = 80; // shift right logical (4 SRL 1 = 2)
byte pc_op_sattra = 81; // return assembler attribute (EQU 4th)
byte pc_op_sattrp = 82; // return program attribute (EQU 5th)
byte pc_op_x2a = 83; // convert hex string to value
byte pc_op_x2b = 84; // convert hex string to binary string
byte pc_op_x2c = 85; // convert hex string to character string
byte pc_op_x2d = 86; // convert hex string to decimal string
byte pc_op_xor = 87; // logical exclusive or (NC)

```

The macro pseudo code instruction fields are as follows:

```

pc_op          - operation code
pc_var_type    - variable or constant operand type
pc_var_loc     - variable location (local, global, or parm)
pc_seta       - integer variable name index or arithmetic constant
pc_setc       - variable name or string constant
pc_sysndx     - current macro instance (used to update local variable on first use)
pc_next       - pointer to next macro pseudo operation in cache
pc_req_opt    - request to optimize pcl list pending (used for initial optimization)

```

The marco pseudo code pointers for each macro source line are maintained using the following arrays:

pcl_start - pointer to first macro pseudo code operation in list for statement
pcl_end - pointer to last macro pseudo code operation in last for statement
pcl_mru_next- next most recently used pcl list pointer
pcl_mru_prev- previous most recently used pcl list pointer
pcl_mru - pointer to most recently used pcl list (last statement executed)
pcl_lru - pointer to least recently used pcl list (released for reuse as required)

See the TRACEP generated macro pseudo code trace file RT\TEST\TESTPC1.TRM and RT\TEST\TESTOPR1.TRM which provide good examples. Below is a very short example program with open macro code loop showing both generated and executed macro pseudo code including SETA PUSHV, ADD, STORV which is optimized to INC on first pseudo code execution:

```

MZ390 PGM INFO MZ390I V1.2.00 Current Date 11/11/06 Time 10:49:24
MZ390 PGM INFO MZ390I Copyright 2006 Automated Software Tools
Corporation
MZ390 PGM INFO MZ390I z390 is licensed under GNU General Public License
MZ390 PGM INFO MZ390I program = D:\WORK\Z390\TEST2.MLC
MZ390 PGM INFO MZ390I options = sysmac(D:\Work\z390\mac+.)
syscpy(D:\Work\z390\mac+.) TRACEP
LOADING FILE D:\WORK\Z390\TEST2.MLC
OPEN CODE 1 1 TEST2 CSECT
OPEN CODE 2 2 BR 14
OPEN CODE 3 &COUNT SETA 0
GEN PC LOC= 1 OP= PUSHA()=0
GEN PC LOC= 2 OP= STORV(&COUNT)=0
OPEN CODE 5 &COUNT SETA &COUNT+1
GEN PC LOC= 3 OP= PUSHV(&COUNT)=0
GEN PC LOC= 4 OP= ADD(0,1)=1
GEN PC LOC= 5 OP= STORV(&COUNT)=1
OPEN CODE 6 AIF (&COUNT GT 2).EXIT
GEN PC LOC= 6 OP= PUSHV(&COUNT)=1
GEN PC LOC= 7 OP= COMPGT(1,2)=0
GEN PC LOC= 8 OP= AIF(0)=.EXIT NO BRANCH
OPEN CODE 7 &ABC SETC 'ABC'
GEN PC LOC= 9 OP= PUSHC()='ABC'
GEN PC LOC= 10 OP= STORV(&ABC)='ABC'
OPEN CODE 8 &BC SETC '&ABC'(2,2)
GEN PC LOC= 11 OP= PUSHV(&ABC)='ABC'
GEN PC LOC= 12 OP= PUSHA()=2
GEN PC LOC= 13 OP= PUSHA()=2
GEN PC LOC= 14 OP= SUBSTR('ABC',2,2)='BC'
GEN PC LOC= 15 OP= STORV(&BC)='BC'
OPEN CODE 9 MNOTE 'COUNT=&COUNT BC=&BC'
OPEN CODE 9 3 MNOTE 'COUNT=1 BC=BC'
  
```

```

OPEN CODE 10      AGO .LOOP
OPEN CODE  5      &COUNT SETA &COUNT+1
EXEC PC LOC= 3 OP= INC(&COUNT)=2
OPEN CODE  6      AIF (&COUNT GT 2).EXIT
EXEC PC LOC= 6 OP= PUSHV(&COUNT)=2
EXEC PC LOC= 7 OP= COMPGT(2,2)=0
EXEC PC LOC= 8 OP= AIF(0)=.EXIT NO BRANCH
OPEN CODE  7      &ABC  SETC 'ABC'
EXEC PC LOC= 9 OP= PUSHC()='ABC'
EXEC PC LOC=10 OP= STORV(&ABC)='ABC'
OPEN CODE  8      &BC   SETC '&ABC'(2,2)
EXEC PC LOC=11 OP= PUSHV(&ABC)='ABC'
EXEC PC LOC=12 OP= PUSHA()=2
EXEC PC LOC=13 OP= PUSHA()=2
EXEC PC LOC=14 OP= SUBSTR('ABC',2,2)='BC'
EXEC PC LOC=15 OP= STORV(&BC)='BC'
OPEN CODE  9      MNOTE 'COUNT=&COUNT BC=&BC'
OPEN CODE  9      4    MNOTE 'COUNT=2 BC=BC'
OPEN CODE  5      &COUNT SETA &COUNT+1
EXEC PC LOC= 3 OP= INC(&COUNT)=3
OPEN CODE  6      AIF (&COUNT GT 2).EXIT
EXEC PC LOC= 6 OP= PUSHV(&COUNT)=3
EXEC PC LOC= 7 OP= COMPGT(3,2)=1
EXEC PC LOC= 8 OP= AIF(1)=.EXIT BRANCH
OPEN CODE 12      5    END
MZ390 PGM INFO   6 * Stats total MLC/MAC loaded = 10
MZ390 PGM INFO   7 * Stats total BAL output lines= 5
MZ390 PGM INFO   8 * Stats total BAL instructions= 3
MZ390 PGM INFO   9 * Stats total macros      = 1
MZ390 PGM INFO  10 * Stats total macro loads   = 1
MZ390 PGM INFO  11 * Stats total macro calls   = 0
MZ390 PGM INFO  12 * Stats total global set names= 41
MZ390 PGM INFO  13 * Stats tot global seta cells = 3
MZ390 PGM INFO  14 * Stats tot global setb cells = 3
MZ390 PGM INFO  15 * Stats tot global setc cells = 35
MZ390 PGM INFO  16 * Stats max local pos parms  = 0
MZ390 PGM INFO  17 * Stats max local key parms  = 0
MZ390 PGM INFO  18 * Stats max local set names  = 8
MZ390 PGM INFO  19 * Stats max local seta cells = 3
MZ390 PGM INFO  20 * Stats max local setb cells = 0
MZ390 PGM INFO  21 * Stats max local setc cells = 5
MZ390 PGM INFO  22 * Stats total array expansions= 0
MZ390 PGM INFO  23 * Stats total Keys          = 920
MZ390 PGM INFO  24 * Stats Key searches       = 997
MZ390 PGM INFO  25 * Stats Key avg comps     = 0
MZ390 PGM INFO  26 * Stats Key max comps     = 2

```

MZ390 PGM INFO 27 * Stats total macro line exec = 19
MZ390 PGM INFO 28 * Stats total pcode line exec = 6
MZ390 PGM INFO 29 * Stats total pcode line gen. = 5
MZ390 PGM INFO 30 * Stats total pcode line reuse= 0
MZ390 PGM INFO 31 * Stats total pcode op gen. = 15
MZ390 PGM INFO 32 * Stats total pcode op exec = 15
MZ390 PGM INFO 33 * Stats total pcode gen opt = 4
MZ390 PGM INFO 34 * Stats total pcode exec opt = 1
MZ390 PGM INFO 35 * Stats total milliseconds = 345
MZ390 PGM INFO 36 * Stats instructions/second = 55