[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 5/00 (2006.01)

[12] 发 明 专 利 说 明 书

专利号 ZL 00805202.6

[45] 授权公告日 2006 年 4 月 19 日

[11] 授权公告号 CN 1252586C

[22] 申请日 2000.3.17 [21] 申请号 00805202.6

[30] 优先权

[32] 1999. 3.19 [33] US [31] 09/273,149

[86] 国际申请 PCT/US2000/007288 2000.3.17

[87] 国际公布 WO2000/057268 英 2000.9.28

[85] 进入国家阶段日期 2001.9.18

[71] 专利权人 联合想象计算机公司

地址 美国纽约州

[72] 发明人 凯文 M·平塔 唐纳德 L·博伦

审查员 胡徐兵

[74] 专利代理机构 中原信达知识产权代理有限责任公司

代理人 关兆辉 谷慧敏

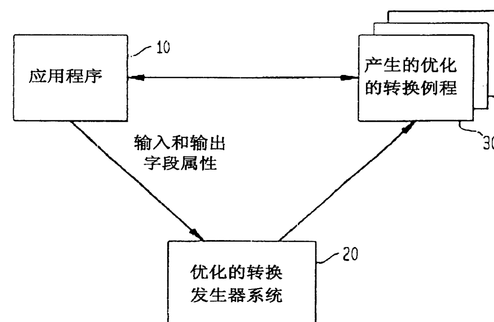
权利要求书 3 页 说明书 7 页 附图 15 页

[54] 发明名称

产生优化的计算机数据字段转换例程

[57] 摘要

一个系统（20）把数据从输入字段类型转换为输出字段类型。系统（20）接收来自一个应用程序（10）的多个输入属性和输出属性，为每组输入属性和输出属性动态地产生多个数据字段转换例程（30），并把多个数据字段转换例程（30）存储在可以被应用程序（10）访问的存储器中。



1. 一种由一个应用程序把多个输入字段类型数据转换为多个输出字段类型数据的方法，所述方法包括：

5 （a）接收一个第一输入字段类型的一个第一属性和一个第一输出字段类型的一个第二属性；

（b）利用所述第一输入字段类型的第一属性和所述第一输出字段类型的第二属性，识别用于将所述第一输入字段类型转换为所述第一输出字段类型的一个第一代码模板；

10 （c）基于所述经识别的第一代码模板，生成一个连续可重复使用的第一优化的转换例程；以及

（d）执行来自所述应用程序的所述连续可重复使用的第一优化的转换例程，以把所述第一输入字段类型转换为所述第一输出字段类型。

15

2. 根据权利要求 1 的方法，其中步骤（d）包括调用来自所述应用程序的所述第一优化的转换例程。

3. 根据权利要求 1 的方法，其中步骤（d）包括同所述应用程序内

20 联地存储所述第一优化的转换程序。

4. 根据权利要求 1 的方法，其中在所述应用程序正在执行时动态地执行步骤（c）。

25 5. 根据权利要求 1 的方法，进一步包括：

（e）接收一个第二输入字段类型的一个第三属性和一个第二输出字段类型的一个第四属性；

（f）利用所述第二输入字段类型的第三属性和所述第二输出字段类型的第四属性，识别用于将所述第三输入字段类型转换为所述第

30 四输出字段类型的一个第二代码模板；

（g）基于所述经识别的第二代码模板，生成一个连续可重复使用的第二优化的转换例程；

（h）执行来自所述应用程序的所述连续可重复使用的第二优化的转换例程，以便把所述第三输入字段类型转换为所述第四输出字段类型。

6. 根据权利要求 1 的方法，其中所述第一和第二属性是字符类型。

7. 根据权利要求 1 的方法，进一步包括为所述第一优化的转换例程产生程序调试方法。

8. 根据权利要求 1 的方法，其中所述建立第一优化转换例程的步骤包括：

设置与所述转换例程相关的一个或多个缺省的过程选项；

构建模板接口块，作为应用程序和代码发生器系统之间的接口；

为所述转换例程获取存储设备；

基于所述第一属性和所述第二属性构建公用字段转换接口块；

利用所述公用字段转换接口块调用所述代码发生器系统，以产生对所述转换例程的代码；以及

在所述存储设备中保存所述转换例程，以备后续使用。

9. 一种把数据从多个输入字段类型转换成多个输出字段类型的方法，所述方法包括：

（a）接收来自一个应用程序的多组输入属性和对应的输出属性；

（b）利用所述的多组输入属性和相应的输出属性，识别用于将所述每种输入属性转换成与其对应的输出属性的多个代码模板；

（c）基于用于将所述每种输入属性转换成与其对应的输出属性的所述多个代码模板，生成多个对应的数据字段转换例程；以及

（d）把所述多个数据字段转换例程存储在所述应用程序可以访问的存储器中。

10. 根据权利要求 9 的方法，其中，所述数据字段转换例程可以由所述应用程序调用。

5 11. 根据权利要求 9 的方法，其中所述数据字段转换例程与所述应用程序内联地存储。

12. 根据权利要求 9 的方法，其中在所述应用程序正在执行时，动态地执行步骤（c）。

10

13. 根据权利要求 9 的方法，其中所述输入和输出属性是字符类型。

14. 根据权利要求 9 的方法，其中所述输入和输出属性是日期类型。

15 15. 根据权利要求 9 的方法，进一步包括为所述多个数据字段转换例程产生程序调试方法。

16. 根据权利要求 9 的方法，进一步包括：

设置与所述的多个转换例程相关的一个或多个缺省的过程选项；

20 构建一个或多个模板接口块，作为应用程序和代码发生器系统之间的接口；

为所述多个转换例程获取存储设备；

基于所述多个输入属性和输出属性构建一个或多个公用字段转换接口块；以及

25 利用所述一个或多个公用字段转换接口块调用所述代码发生器系统，以产生对所述一个或多个转换例程的代码。

产生优化的计算机数据字段转换例程

5

发明领域

本发明针对计算机数据。更具体地，本发明针对一种类型的计算机数据字段向另一种类型的转换。

发明背景

10

在许多情况下，在计算机进行信息处理过程中，计算机数据必须从一种数据字段类型转换成另一种。例如，当数据从一个程序被传送到另一个程序时，数据通常在此过程中要经历几次转换，比如从文本数字转换为一个二进制数。

15

用于转换数据的典型技术包括使用一个类属数据转换例程。当必须转换数据的整个记录时，转换例程必须确定记录中的每个数据字段的特性或属性是什么。这可能要求转换例程为每个记录的每个字段执行相同的决策树，尽管每个字段具有在逐行的基础上不变化的已知特性。因此，许多计算机周期被浪费在就每个数据字段再三问比如“本字段类型是字符、整数、等等？”问题上。

20

基于上述内容，需要有一种提供数据字段的有效转换的系统。

发明内容

25

本发明的一个实施例是一个用于把数据从输入字段类型转换为输出字段类型的系统。该系统从一个应用程序中接收多个输入属性和输出属性，动态地为每组输入属性和输出属性产生多个数据字段转换例程，并把多个数据字段转换例程存储在应用程序可以访问的存储器中。

30

附图简介

图 1 是表示根据本发明的一个实施例的一个优化的转换发生器系统的功能性的总揽方框图。

图 2 是由根据本发明的一个实施例的系统执行用来产生优化的转换例程的步骤的流程图。

图 3 是当使用例程把输入字段转换为输出字段时由应用程序执行的步骤的流程图。

图 4 是当应用程序调用时，由转换发生器系统执行的代码产生步骤的流程图。

图 5a 和 5b 表示在本发明的一个实施例中使用的动态代码构建的一般例子。

图 6a-6h 表示执行 CHARACTER 到 CHARACTER 转换的一个动态代码产生例程的特例。

详细说明

本发明的一个实施例是为一个应用程序所要求的每种转换类型产生优化的数据字段向数据字段的转换例程的系统。图 1 是表示根据本发明的一个实施例的一个优化的转换发生器系统 20 的功能性的总揽方框图。系统 20 可以在软件中实施，并可以在一个包括中央处理单元和存储器的通用计算机上执行。在一个实施例中，用 IBM/360 机器指令实施系统 20。

一个应用程序 10 需要执行一种或者更多种字段转换类型。对于每一种转换类型，应用程序 10 为系统 20 提供输入（或者“源”）和输出（或者“目的”）字段属性。对于每组输入和输出字段属性，系统 20 动态地产生一个执行转换的优化的转换例程 30。优化的例程 30 被置于应用程序 10 可用的存储器中。

一个实施例中的例程 30 被作为独立例程产生，该独立例程能够被连续地再用，并在当需要一个转换时，由应用程序 10 使用，例如，

一个应用程序接口（“API”），来调用。在另一个实施例中，例程 30 被作为代码程序块（code chunk）来产生，这些代码程序块被插入应用程序 10 中与其内联，当需要一个转换时可以被直接访问。

5 本发明的一个优点是通过构建专门适合于输入和输出字段属性的优化的转换例程，例程的每次执行省却许多通常需要的在每次执行转换时识别字段属性的指令。

图 2 是根据本发明的一个实施例由系统 20 执行用来产生优化的
10 转换例程 30 的步骤的流程图。应用程序 10 在步骤 100 确定了输入字段和输出字段具有什么属性之后，由系统 20 执行步骤。

在步骤 102，系统 20 设置产生的转换例程 30 的缺省过程选项。这些选项可能包括产生的转换例程 30 是否是可调用函数（即，能由
15 应用程序 10 调用），或者是否可以内联复制到应用程序 10 中。步骤 102 构建一个模板接口块 104，其为应用程序 10 和转换发生器系统 20 之间的一个接口。步骤 102 还产生一个获得需要的存储空间和进行错误检查的启动调用 106。

20 在步骤 108，启动一个循环，直到穷举了所有必须被转换的字段，循环才结束。

在循环过程中，在步骤 110，从应用程序 10 接收每组输入和输出属性。通过一个 API 来接收属性，而且步骤 110 还基于这些属性，
25 构建一个公用字段转换接口块 116。

在步骤 112，使用公用接口块 116，调用系统 20 的代码发生器。步骤 112 产生代码 118。

30 在步骤 114，保存一个指向产生的字段转换例程 30 的函数指针。

图 3 是当使用例程 30 把输入字段转换为输出字段时由应用程序 10 执行的步骤的流程图。

5 在步骤 122 中，应用程序正在进行处理。在步骤 124，应用程序获得待转换的源或者输入数据。典型地，步骤 124 涉及读取一个或者多个记录。

在步骤 126，为每个读取的记录启动一个循环。在步骤 128，在
10 一个实施例中调用用于转换的适当转换例程 30。

当所有数据字段和记录被转换时，在步骤 132，要求代码发生器系统 20 终止。其结果是在步骤 134 释放了存储空间。

15 在步骤 136，应用程序 10 继续处理。最后，在步骤 138，应用程序 10 结束。

图 4 是当应用程序 10 调用时，由转换发生器系统 20 执行以产生代码的代码产生步骤的流程图。

20

在步骤 200，系统 20 通过，例如，建立需要的存储空间，检查无效选项和指定应如何产生代码来进行初始化。

在步骤 202，系统 20 验证特定的字段转换选项，例如检验输入
25 和输出长度是正确的。步骤 202 还确定当代码产生时代码有多大。如果产生的代码将被内联存储，应用程序 10 可以使用上述信息。

在步骤 204，系统 20 使用字段转换接口块 116 来构建转换例程。

30 在步骤 206，在步骤 200 获得的存储空间被释放。

步骤 202 和 204 经历同样的内部过程。因此，在步骤 208，确定了输入字段类型。输入字段类型的例子包括字符输入 210 或者专用时间格式输入 212。但是，本发明支持任何输入字段类型。

5

类似地，在步骤 214，确定输出字段类型。输出字段类型的例子也包括字符输入 213 或者专用时间格式输入 215，但是本发明支持任何输出字段类型。

10 在步骤 216，如果执行步骤 202，确定产生的代码的大小。在步骤 218，如果执行步骤 204，产生字段转换例程 30。

如所公开的，根据本发明的一个实施例的系统 20 为每组输入和输出字段属性动态地产生优化的转换例程 30。接着，由应用程序 10 利用例程 30 处理转换。输入和输出字段被系统 20 归类为原型数据类型，每种数据类型都有可定义的属性和转换行为。例如：

15

- 字符数据类型将是一个具有最大长度属性和 CCSID（或字符设置代码页）属性的定长字段。
- 日期数据类型将是一个具有最大长度属性和确定日期中使用的分隔符的位置和类型的格式属性（ISO，EUR，等）的定长字段。

20

在优化的转换发生器系统 20 的一个实施例中包括的一些以前说明的或者附加的特征包括：

- 可选地获得和释放用于 API 控制块和/或产生的代码的存储空间。

25

- API 控制块可以由 API 管理函数来链接和模板化。
- 可以通过使用一个宏接口来构建 API 控制块。
- 转换例程可以利用寄存器来直接寻址输入和输出字段的位置。寄存器可以由应用程序 10 通过 API 参数来选择。

30

- 基于 API 参数，在转换之后源字段地址寄存器可以可选地被

递增到输入字段的末尾。

- 基于 API 参数，在转换之后目的字段地址寄存器可以可选地被递增到格式化字段的末尾。
- 基于 API 参数，一个附加寄存器可以被递增所转换字段的长度。
- 基于 API 参数，可以产生用于转换例程的标准连接。
- 基于 API 参数，可以指定转换错误退出（Conversion Error exits）来处理枚举的转换错误条件。
- 基于 API 参数，可以产生字符代码集翻译转换代码（即，ASCII 字符字段可以被翻译成 EBCDIC 字符字段）。
- 可以产生转换例程来利用由正在为之产生代码的操作系统级支持的最新指令。

在一个实施例中，系统 20 基于 API 控制块中的各种设置，通过在调用应用程序 10 可以访问的存储空间中构建代码程序块，动态地产生代码。如结合流程图中的讨论，产生代码涉及以下步骤：

1. 为代码获得存储空间。
2. 识别所需要的代码模板。
3. 移动代码模板。
4. 修改代码模板。
5. 向调用应用程序返回可执行代码。

进一步，在一个实施例中，基于 API 规范，系统 20 可以为动态产生的代码可选地产生程序调试方法。该方法可以包括一个可选动态分配的输出文件，对于每个字段转换，其包括：API 选项和通用处理选项的报告，API 选项被用于每个动态产生的例程，可以用该例程来确保字段属性的正确性；和动态产生的例程的一个分解的列表，分解的列表由系统 20 内的一个内部分解器提供，列表可以被用来识别转换代码的不精确性和进一步优化的区域，并有助于解决产生的代码故障。

图 5a 和 5b 表示在本发明的一个实施例中使用的动态代码构建的一般例子。

5 图 6a-6h 表示执行 CHARACTER 到 CHARACTER 转换的一个动态代码产生例程的特例。

在此专门解释和/或说明了本发明的几个实施例。但是，应当理解本发明的修改和变动为上述教导所函盖，并且置于所附权利要求的

10 权限之内，而不背离本发明的精神和旨在范围。

图1

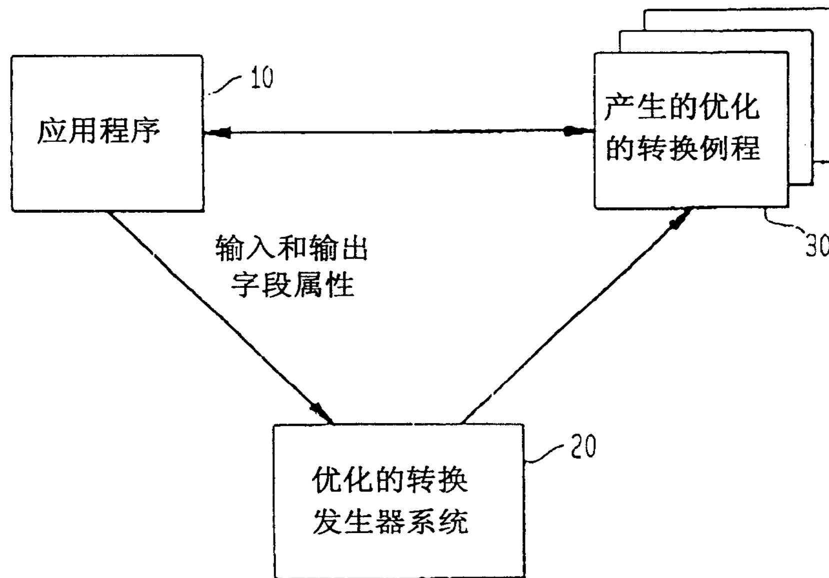


图2

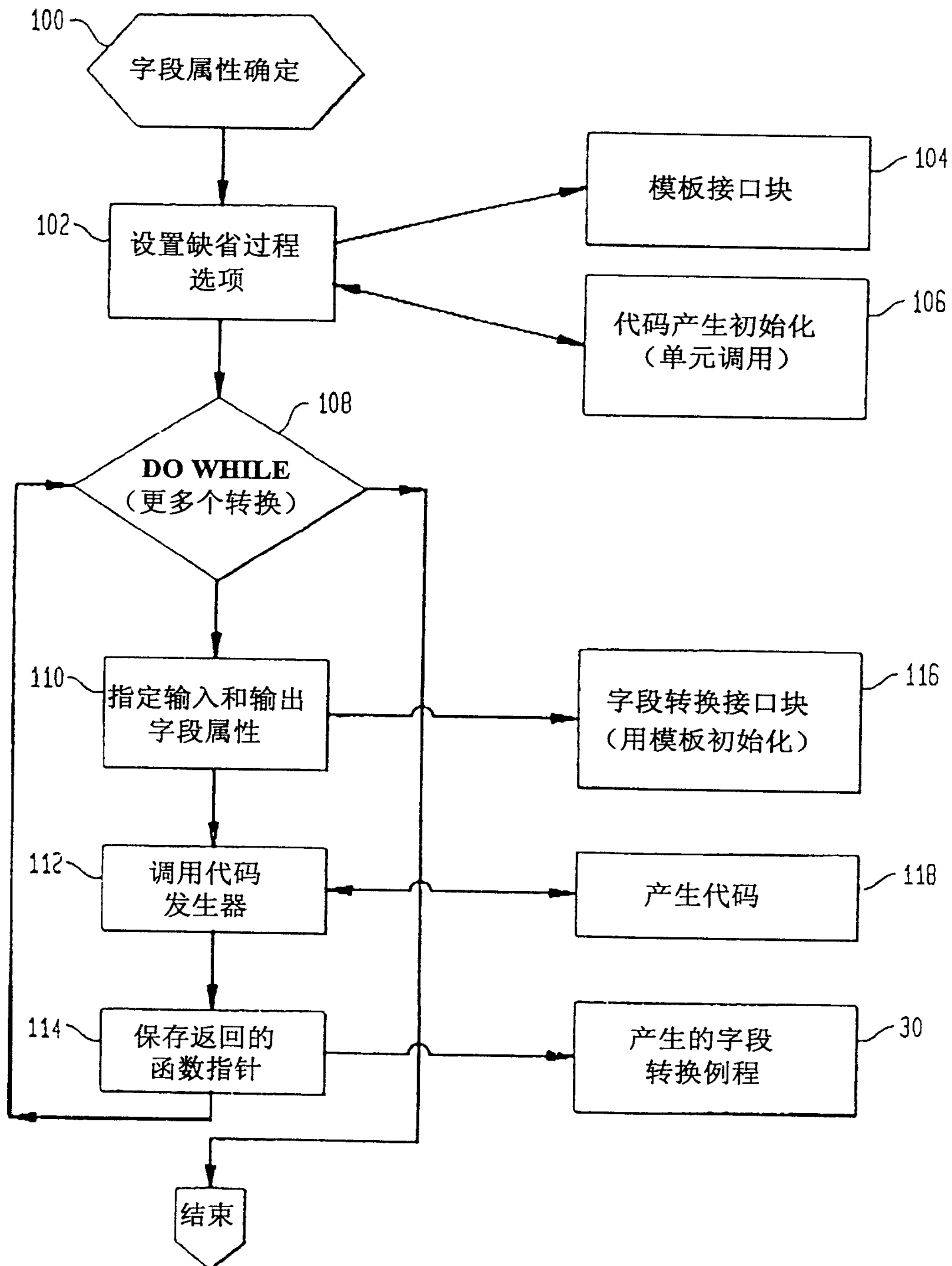


图3

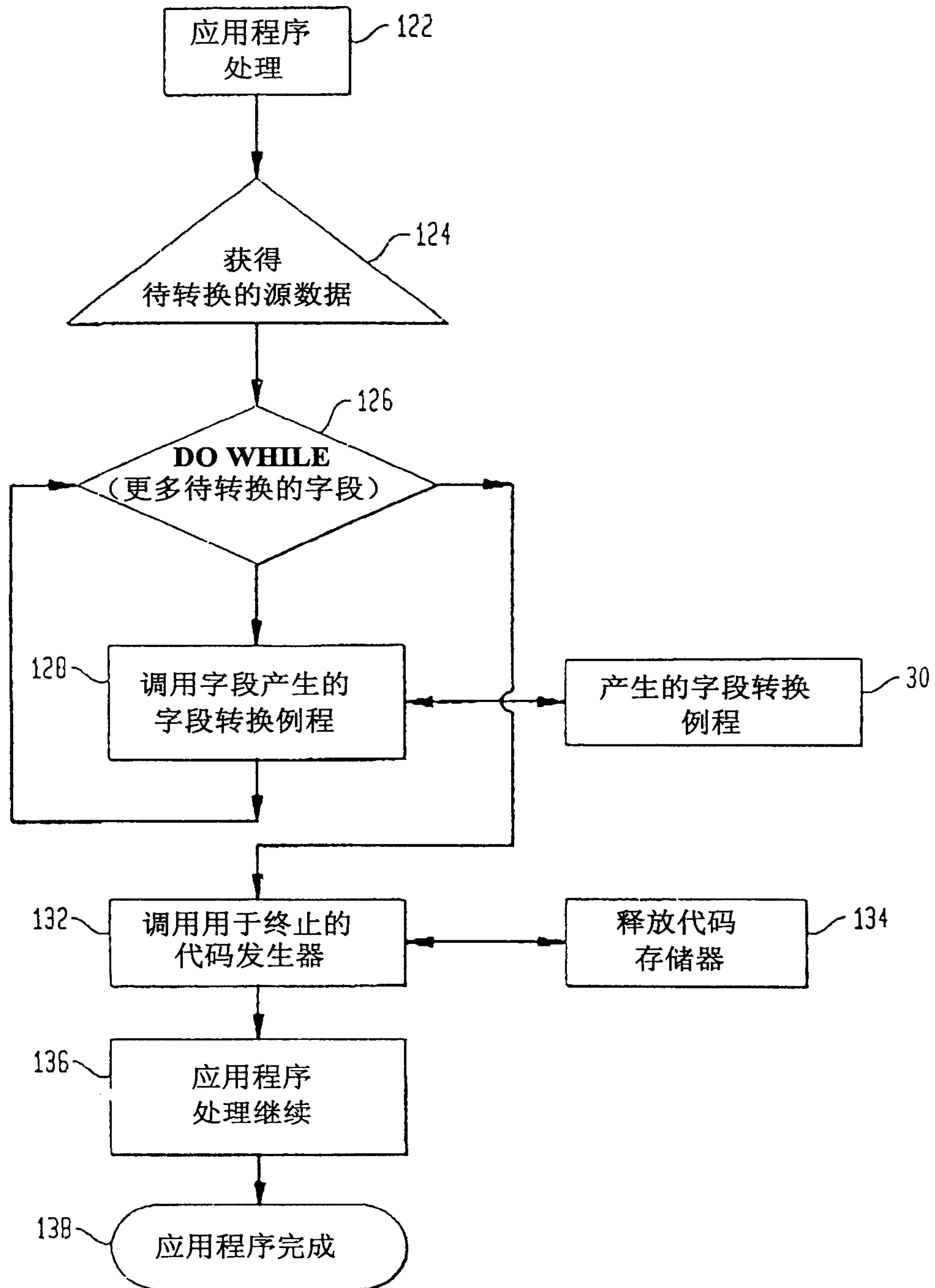


图4A

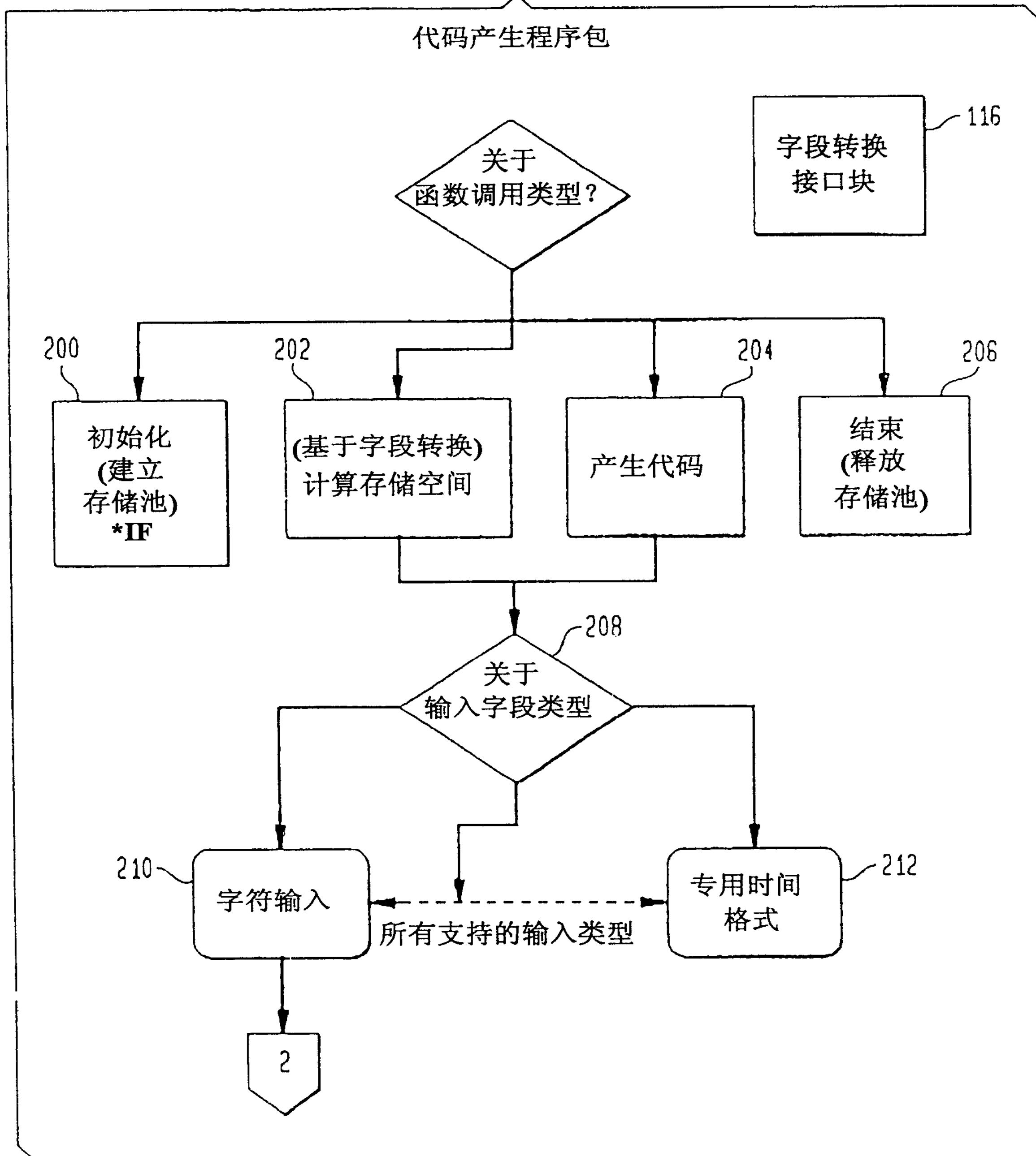
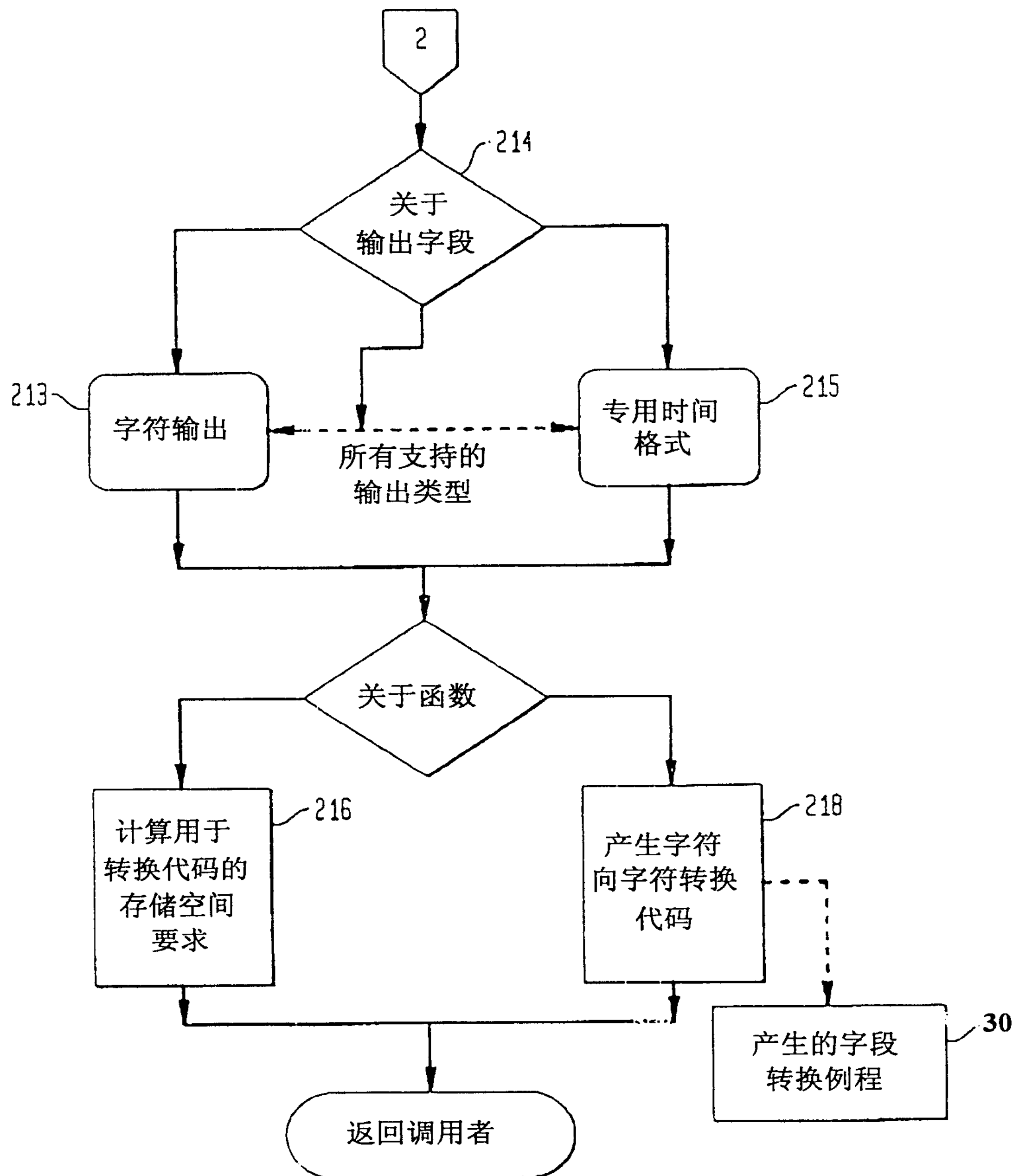


图4B



```
R5      =      Current Instruction Offset within application buffer
R6      =      Current Instruction Address within application buffer
R7      =      Work Register - used for calculating offsets, etc
R12     =      Base register of code generator and template code

SLR            R5,R5                    clear offset
L              R6,$BCB_BCODE_@          get address of user buffer

* if linkage required call standard linkage builder
IF (TM,$BCB_PFLAG1,$BCB_LINKAGE,O)
   SETF    LINKAGE
   IF (CLI,$BCB_LINKAGE_TYPE,EQ,C'N')
      RESETF  LINKAGE
   COND ELSE
* call standard linkage builder
      #BAS    14,=A(BURST_ENTRY_LINKAGE)
   ENDIF
ELSE
   RESETF  LINKAGE
ENDIF
****
STDRETURN           -        RETURN TO APPLICATION
* $BCB_BCODE_@ WILL POINT TO BUILT CODE
****

*
* Routine to build standard entry linkage
*
BURST_ENTRY_LINKAGE CSMSUBI BASE=R10,WORKREG=R3
*
* Move Template code into user buffer
       MVC    0(STD_ENTL_010_L,R6),STD_ENTL_010
*
* Modify  " LA      R14,0(0)" instruction
* Get Offset to Savearea using equate STD_ENTL_010_SA_A
* Set base register for instruction to R12
* Set D(X,B) of instruction (R7 contains constructed D(X,B))
       LA     R7,STD_ENTL_010_SA_A(,R5)
       O      R7,=X'0000C000'
       STH    R7,STD_ENTL_010_SA_T(,R6)
*
```

图5a

```
* Modify  " B      0(R12)" instruction
* Get offset of branch target using equate STD_ENTL_010_B_A_T
* Set D(X,B) of instruction (R7 contains constructed D(X,B))
* ** Note X (index register) has been set by assembler as R12
*    STH does not change the instruction's index register
       LA     R7,STD_ENTL_010_B_A_T(,R5) CALC OFFSET FOR BRANCH TARGET
       STH    R7,STD_ENTL_010_B_A(,R6)    SET BRANCH D(X,B)
*
* Increment Next Instruction Offset (in R5) by length of code
* Increment Next Instruction Address (in R6) by length of code
       LA     R5,STD_ENTL_010_L(,R5)
       LA     R6,STD_ENTL_010_L(,R6)
*
* Return to caller
* Code has been built and the Instruction Offset and Address registers
* have been updated for next instruction construction

           CSMSUBO
*- STANDARD ENTRY LINKAGE ------------------------------------------|
*                                                                  |
*------------------------------------------------------------------|
STD_ENTL_010 DS  0S                                                |
            STM     R14,R12,12(R13)                                |
STD_ENTL_010_SA_T EQU  *-STD_ENTL_010+2                            |
            LA      R14,0(0)                BURSTED SAVEAREA+0     |
            ST      R13,4(,R14)                                    |
            ST      R14,8(,R13)                                    |
            LR      R13,R14                 R13 = BURSTED SAVEAREA |
            LR      R12,R15                 SET BURSTED BASE REG   |
STD_ENTL_010_B_A  EQU  *-STD_ENTL_010+2                            |
            B       0(R12)                  WS_BRANCH              |
STD_ENTL_010_SA_A EQU  *-STD_ENTL_010                              |
            DC      18F'0'                                         |
STD_ENTL_010_B_A_T EQU  *-STD_ENTL_010                             |
STD_ENTL_010_L   EQU   *-STD_ENTL_010                              |
*------------------------------------------------------------------|
```

图5b

```
* Call made by API passing API $BURSTCB control block
* Control block contains field attributes and conversion
* options

* Reset processing flags
*  NO_BUILD -> doing conversion routine storage calculation
*  CALLED_ROUTINE -> creating a called routine
* Check for API block -> if not there abend with dump
* Copy passed API block to working storage (IN_BCB)

MAIN_0000 DS      0S
*
          RESETF  NO_BUILD
          RESETF  CALLED_ROUTINE
*
          LTR     R1,R1
          BNZ     MAIN_0005
*
          ABEND   001,DUMP
*
MAIN_0005 DS      0S
          MVC     IN_BCB($BCB_LENGTH),0(R1)
*
          LA      R9,IN_BCB              R9 = ADDRESS OF $BURSTCB
          USING   $BURSTCB,R9
*
* If calculate storage requested SET NO_BUILD
          IF (CLC,$BCB_FUNC,EQ,=Y($BCB_CALC_STORAGE))
             SETF    NO_BUILD
          ENDIF

* INITIALIZE WORKING STORAGE
*  If actually BUILDING code (not NO_BUILD)
*    1. Obtain offset from beginning of BASE REGISTER
*    for code.  If callable routine this has been set to 0.
*    otherwise this we are building inline code within the application's
*    user managed buffer and the offset will set to current instruction offset
*    within the buffer.
*     2. Obtain address of passed code buffer
*     3. Calculate current instruction address based on offset into buffer
```

图6a

```
MAIN_STRT DS       0S
          IF (¬NO_BUILD)
             LH       R5,$BCB_BCODE_OFFSET
             L        R6,$BCB_BCODE_@
             LA       R6,0(R5,R6)
          ELSE
             SLR      R5,R5                    CLEAR FOR ACCUM
             SLR      R6,R6                    CLEAR FOR ACCUM
          ENDIF
*
* INITIALIZE WORK FIELDS FOR ANY COLUMN CONVERSION
* 1. Obtain input field's addressing register
* 2. Build RX type assembler instruction D(X,B) with offset 0
* 3. Obtain output field's addressing register
* 4. Build RX type assembler instruction D(X,B) with offset 0
*    set template for output D(X,B)
* 5. Obtain input and output lengths
* 6. Set Current working D(X,B) templates
          SLR      R7,R7
          ICM      R7,B'0001',$BCB_IREG
          SLL      R7,4                              SHIFT NIBBLE
          STC      R7,WB_INIT_SOURCE_DB
          ICM      R7,B'0001',$BCB_OREG
          SLL      R7,4                              SHIFT NIBBLE
          STC      R7,WB_INIT_TARGET_DB
          MVC      WB_TOT_INPUT_LEN,$BCB_ILEN
          MVC      WB_TOT_OUTPUT_LEN,$BCB_OLEN
          MVC      WB_SOURCE_DB,WB_INIT_SOURCE_DB    RESET DB
          MVC      WB_TARGET_DB,WB_INIT_TARGET_DB    RESET DB
*
* CHECK FOR LINKAGE REQUIREMENTS
* IF LINKAGE = E (BASIC ENTRY - SAVE/RESTORE R14) THEN
*    BURST_WORK_BRANCH WILL SAVE R14 AND SET RESTORE_R14
*    BURST_EXIT_LINKAGE RESTORES R14 AND BASR R14
* ENDIF
          RESETF   RESTORE_R14
          IF (TM,$BCB_PFLAG1,$BCB_LINKAGE,O)
             SETF     LINKAGE
             IF (CLI,$BCB_LINKAGE_TYPE,EQ,C'N')
                RESETF  LINKAGE
             COND ELSE
                #BAS     14,=A(BURST_ENTRY_LINKAGE)
             ENDIF
          ELSE
             RESETF  LINKAGE
          ENDIF
```

图6b

```
* CALL INPUT TYPE PROCESSING ROUTINE
* 1. Get address of input field type table
*    This table contains an index of supported input types
*    with their associated code generation routines
* 2. Call code generation routine for Input field type
*    In this case INPUT FIELD TYPE IS CHARACTER
*    INPUT FIELD TYPE CHARACTER calls routine named CHARACTER
**** Further down subroutine CHARACTER is shown
          L        R14,=A(TYPE_TABLE)
          LH       R15,$BCB_ITYPE
          LA       R15,0(R14,R15)
          L        R15,0(,R15)
          BASR     R14,R15

* Subroutine has built conversion code for INPUT TYPE CHARACTER and OUTPUT TYPE CHARACTER
* Check for other process options such as: accumulate a source addressing register,
* accumulate a target addressing register, or accumulate alternate register.
* alternate register usually is a total output length accumulator used by the calling
* application to keep track of an aggregate of all output field lengths
* 1. IF source addressing register accumulate requested build code to accumulate
* 2. IF target addressing register accumulate requested build code to accumulate
* 3. IF length register accumulate requested build code to accumulate
* 4. IF exit linkage requested build exit linkage
* 5. RETURN TO API CALLER with generated conversion routine
MAIN_0200 DS       0S
          IF (TM,$BCB_PFLAG1,$BCB_SRC_ACUM,O)
             LH       R0,WB_SOURCE_ACCUM_INDEX
             IC       R1,$BCB_SRC_ACUM_REG
             LH       R7,WB_TOT_INPUT_LEN
             #BAS     14,=A(FIXED_ACCUM)
          ENDIF
*
          IF (TM,$BCB_PFLAG1,$BCB_TRG_ACUM,O)
             LH       R0,WB_TARGET_ACCUM_INDEX
             IC       R1,$BCB_TRG_ACUM_REG
             LH       R7,WB_TOT_OUTPUT_LEN
             #BAS     14,=A(FIXED_ACCUM)
          ENDIF
*
          IF (TM,$BCB_PFLAG1,$BCB_TRG_L_ACUM,O)
             LH       R0,WB_TARGET_ACCUM_INDEX
             IC       R1,$BCB_TLN_ACUM_REG
             LH       R7,WB_TOT_OUTPUT_LEN
             #BAS     14,=A(FIXED_ACCUM)
          ENDIF
*
* BURST EXIT LINKAGE
          IF (LINKAGE)
             SETF       CLEAR_R15
             #BAS       14,=A(BURST_EXIT_LINKAGE)
          ENDIF
          RETURN to CALLER
```

图6c

```
*-------------------------------------------------------------------*
* Character Input Field Type Conversion Routine                    *
* Abstract:                                                         *
*     This routine is called to either build Character Input        *
*     Fields to all supported Output Field Types, or to calculate   *
*     storage requirements for generated conversion routines for    *
*     Input field type Character                                    *
*                                                                   *
* CHARACTER field type constraints                                  *
*    These field types will be of fixed length                      *
*    Maximum length is 254 8bit bytes                               *
*    They may be proceeded with a null field indicator of length    *
*       1 byte that will contain values of x'00' for non-null fields *
*       and x'ff' for nulled fields.  Nulled fields will not be     *
*       converted accept to indicate on output that field was null  *
*    There values are of EBCDIC CCSID (character code set) unless   *
*    a CCSID is specified through the API.                          *
*                                                                   *
*-------------------------------------------------------------------*


CHARACTER CSMSUBI BASE=R10,WORKREG=R3
* Use branch table generated by API to branch on output type (BTYPE=O)
* Example is demonstrating character to character conversion
* Branch will be taken to CHAR_CHAR_0000
         L        R15,=A(RET_RC_32)
         $BURST   BTABLE,                                          X
               BREG=1,                                             X
               BTYPE=O,                                            X
               UNSUPPORTED=0(,R15),                                X
               CHAR=CHAR_CHAR_0000,                                X
               LVARC=CHAR_VARC_0000,                               X
               VARC=CHAR_VARC_0000

*-@PSEUDO-CODE@-----------------------------------------------------
*                CHARACTER TO CHARACTER CONVERSION
*
* - DETERMINE WORKING STORAGE
*   Some conversions require the generation of local working storage
*   Working storage is generated according to specific conversion options and
*   specific input and output field attributes to avoid generating more storage
*   than needed.
*
*   IF CONVERTING CCSID'S (character code sets) THEN
*     IF using a character translation table (uses TR instruction)
*       Build BRANCH over working storage
*       Build FULL WORD to hold Address character translation table
*       UPDATE Previously built Branch instruction to branch to current offset
*          (offset is next halfword aligned byte where next instruction is to be built)
*      ENDIF
*   ENDIF
*
```

图6d

```
*   IF INPUT LENGTH is GREATER than OUTPUT LENGTH
*      current implementation allows for truncation of trailing spaces
*      If input field being converted by generated code contains non-spaces
*         that won't fit into output field of lesser length then conversion
*         error 4 routine will be called to return a value of 4 in R15
*
*      1. Build BRANCH over working storage
*      2. Build a buffer full of spaces to be used in INPUT field compare
*      3. Build Conversion error routine to return error #4
*      4. UPDATE Previously built Branch instruction to branch to current offset
*          (offset is next halfword aligned byte where next instruction is to be built)
*    ENDIF
* - DETERMINE WORKING STORAGE
*
*-@PSEUDO-CODE@--------------------------------------------------------CHAR_CHAR_0000 DS 0S
*
* BURST WORKAREA IF CONVERSION ERROR OR CONVERT CCSID
          TM      $BCB_PFLAG2,$BCB_CCSID_CNV
          BNZ     CHAR_CHAR_0020
          CLC     $BCB_ILEN,$BCB_OLEN
          BNH     CHAR_CHAR_0040
*
CHAR_CHAR_0020 DS 0S
          #BAS    14,=A(BURST_WORK_BRANCH)
*
          IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV,NZ)
            IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV_ATOE,O)
               #BAS    14,=A(BURST_BWK_TO_E_XLATE_@)
            ELSE
               #BAS    14,=A(BURST_BWK_TO_O_XLATE_@)
            ENDIF
            #BAS    14,=A(BURST_BWK_FULL)
            STH     R7,WB_SAVE_R2_OFFSET
          ENDIF
*
* IF ILEN > OLEN THEN NEED FOLLOWING WORK FIELDS
*    BURST BUFFER255 - SPACES
*    BURST #@ERROR4 CALL
* ENDIF
          IF (CLC,$BCB_ILEN,GT,$BCB_OLEN)
             #BAS    14,=A(BURST_BWK_BUFFER255)
*
*
             LA      R1,4                          #@ERROR4
             #BAS    14,=A(BUILD_CNVERR)
          ENDIF
*
          #BAS    14,=A(UPDATE_WORK_BRANCH)
*
*
```

图6e

```
* IF OUTPUT NULLABLE THEN
*   BURST MOVEMENT OF NULL INDICATOR
*   R1 = X'00' FOR MVI Instruction Builder
*   WB_TARGET_DB (current target D(B)) USED FOR INDICATOR LOCATION
*   Build MVI OF NULL INDICATOR (MVI_0000)
*   UPDATE Current TARGET D(B) TO ALLOW DATA TO SKIP NULL INDICATOR
*   ADD 1 TO TOT OUTPUT LENGTH (FOR NULL INDC) (this allows for accumulation requests)
* ENDIF
CHAR_CHAR_0040 DS 0S
          IF (TM,$BCB_OFLAG1,$BCB_ONULL,O)
             SLR    R1,R1                          CLEAR SOURCE BYTE
             #BAS   14,=A(MVI_0000)                BURST MVI NULL INDC
*
             LH     R1,WB_TARGET_DB                UPDATE TARGET DB
             LA     R1,1(,R1)
             STH    R1,WB_TARGET_DB
*
             LH     R1,WB_TOT_OUTPUT_LEN           UPDATE OUTPUT LEN
             LA     R1,1(,R1)
             STH    R1,WB_TOT_OUTPUT_LEN
          ENDIF
*
*  IF input length < then output length
*     call routine to build code to pad output field with spaces
*  ELSE
*    IF input length = Output length
*      Call routine to build an MVC instruction
*          This routine uses current source and target D(B)'s
*           and the output length to construct the instruction
*    ELSE
*      input length > output length
*      Call routine to build an MVC instruction
*           This routine call will use the input length (since it shorter)
*           (source and target D(B)'s will be used
*       Build Code to check for truncation of only spaces
*    ENDIF
*   ENDIF
          LH     R1,$BCB_ILEN                   GET INPUT LEN
          LH     R2,$BCB_OLEN                   GET OUTPUT LEN
*
          CR     R1,R2                          CHECK LENGTHS
          BE     CHAR_CHAR_0050                    EQUAL
          BH     CHAR_CHAR_0100                    I > O ->
*
* INPUT LENGTH LESS THAN OUTPUT -> NEED TO PAD
* Build Character padding code
          #BAS   14,=A(SSP_0000)
*
* Build code TO MOVE CHARACTER FIELD TO CHARACTER FIELD
CHAR_CHAR_0050 DS     0S
          #BAS   14,=A(MVC_0000)                BURST MVC INSTRUCTION
          B      CHAR_CHAR_0200
*
```

图6f

```
* INPUT field is too large to fit
* Build code TO MOVE CHARACTER FIELD TO CHARACTER FIELD using input field's length
CHAR_CHAR_0100 DS      0S
          LR      R1,R2
          #BAS    14,=A(MVC_0000)              BURST MVC INSTRUCTION
*
* MOVE CHECK FOR SPACES
* IF TRUNCATED DATA NOT SPACES THEN #@ERROR4

          IF  (¬NO_BUILD)
*
          MVC     0(CHAR_CHAR_010_L,R6),CHAR_CHAR_010
*
* SET LENGTH OF COMPARE
          LH      R7,$BCB_ILEN
          SR      R7,R1
          BCTR    R7,0
          STC     R7,CHAR_CHAR_010_OLEN_A(,R6)
*
* SET SOURCE DB TO SOURCE + OLEN-1
          LH      R7,WB_SOURCE_DB
          LA      R7,0(R1,R7)
          BCTR    R7,0
          STH     R7,CHAR_CHAR_010_SDBN_A(,R6)
*
* UPDATE BUFFER OFFSET
          LH      R7,WB_BUFFER255_OFFSET
          O       R7,=X'0000C000'
          STH     R7,CHAR_CHAR_010_B255_A(,R6)
*
* UPDATE #@ERROR4 BRANCH
          LH      R7,WB_CNVERR4_OFFSET
          STH     R7,CHAR_CHAR_010_BERR_A(,R6)
*
          ENDIF             (NO_BUILD)
*
          LA      R5,CHAR_CHAR_010_L(,R5)
          LA      R6,CHAR_CHAR_010_L(,R6)
*
```

图6g

```
* CHECK FOR TRANSLATION of CCSID's
* If translation requested call translation routine generator
*   *** note translation routine will perform accumulation
*       operation if API requested it.  If accumulation is performed
*       by the routine the IN_BCB (copy of API block used by generator)
*       will be updated to turn off accumulation by the main process
*       done upon CHARACTER subroutine (see above)
CHAR_CHAR_0200 DS     0S
            IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV,NZ)
*                IF IREG =2 AND SRC_ACCUM TR INST WILL BUMP REG
                 SETF   SAVE_R2
                 IF (CLC,$BCB_IREG,EQ,=H'2'),AND,                         X
                (TM,$BCB_PFLAG1,$BCB_TRG_ACUM+$BCB_TRG_L_ACUM,NZ)
                    RESETF SAVE_R2
                    NI     $BCB_PFLAG1,X'FF'-$BCB_SRC_ACUM
                 ENDIF
                 RESETF XLATE_TO_E
                 IF (TM,$BCB_PFLAG2,$BCB_CCSID_CNV_ATOE,O)
                    SETF XLATE_TO_E
                 ENDIF
                 #BAS   14,=A(DO_XTAB_SHORT)
            ENDIF
*
CHAR_9999 DS     0S
          B      CHARACTER_END
*-------------------------------------------------------------------|
* BURST CHARACTER TO CHARACTER ILEN > OLEN
* TEMPLATE CODE USED FOR NON-SPACE TRUNCATION                       |
*-------------------------------------------------------------------|
CHAR_CHAR_010 DS 0S                                                 |
CHAR_CHAR_010_OLEN_A EQU *-CHAR_CHAR_010+1    LEN OF CLC            |
CHAR_CHAR_010_SDBN_A EQU *-CHAR_CHAR_010+2    LOC OF SOURCE TO COMP |
CHAR_CHAR_010_B255_A EQU *-CHAR_CHAR_010+4    LOC OF 255 SPACES     |
          CLC    0(0,0),0(0)                  SDB+(OLEN-1),BWK_BUFF255 |
CHAR_CHAR_010_BERR_A EQU *-CHAR_CHAR_010+2                          |
          BNE    0(R12)                       NOT SPACES? -> #@ERROR4 |
CHAR_CHAR_010_L   EQU *-CHAR_CHAR_010                               |
*-------------------------------------------------------------------|
```

图6h