

484114

NOV 11 1981

TR58 - 0040

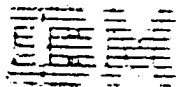
November 1, 1981

MICROCODE DEVELOPMENT
USING DESIGN LANGUAGE

M. Nishihashi, N F C P

82A000023

TECHNICAL REPORT



Fujisawa Development Laboratory

MICROCODE DEVELOPMENT
USING DESIGN LANGUAGE

Mikitoshi Nishihashi

(Non-Formatted Communication
Product)

Background

Generally, the needs to improve the development methods of the software are strong and their realization is getting to indicate the trends to earlier development phase.

In the microcode development of a display, one attempt replaced with the Conventional method is made at some general design stage. That is, by progressing the detail design using a PL/I-like design language, following objectives are to be reached:

- (1) To save the building and maintenance of flow chart in order to facilitate the design, coding and maintenance.
- (2) To search the ability of means for vendorization of microcode development.

Process Overview

This attempt was started at the time when the detail design had been entered after completion of general design. We had started the detail design using the flow chart, but we stopped it and started the detail design using the PL/I -like design language according to the following procedure:*

- (1) First, determined the syntax as required,
- (2) Wrote the contents of the general design according to the (1) above,
- (3) Broke them down to the appropriate stage of details through the edit with the CMS and desk work to obtain the detail design,
- (4) Compiled them manually to obtain the M6800 source code.**
- (5) Then, the debug work was progressed.

The processed (3) to (5) are felt the consistent work between the display terminal and the desk. According to the above process, we succeeded the developing of the microcode faster than the expected.***

Note: *) Therefore, this is not the experience to progress the detail design using only the "pure" design

language excluding the flow chart. The image obtained by the flow chart written by the contents of the initial design seems to be some helps to the design performed by the design language afterwards. We will discuss this later.

- ***) For the M6800, we compiled manually because of the lack of the compiler, however, if a compiler exists, as in the case of WOOK STOCK-II, etc., we would use it.
- ***) Since some other works intervened or the development speed was limited due to other factors than the microcode, the quantitative data were not calculated or obtained because of its meaninglessness.

Design Language and Pseudo Code

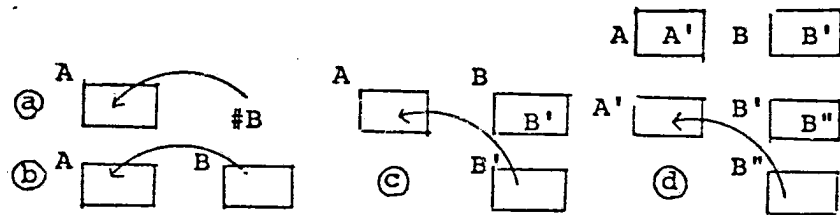
Here lists up the general part of the syntax of the design language being used. The appropriate level of the formalization can be obtained by the addition or deletion of the regulation according to the user's environment.

(1) Punctuation : ; Address: , Sentence;

- (2) Sentence Group Do; ----- END;
- (3) Branch and Jump of Control IF THEN ELSE, GO TO, CALL,
RETURN FROM SUBROUTINE
- (4) Logical Connective \neg, \wedge, \vee
- (5) Relative Expression (in if caluse) $=, \neg =, >, <,
>=, <=$
- (6) Substituting Sentence

The "=" used in rather than "if caluse" means that the value at the right side is subsituted for the memory area at the left side. In this case, "()" indicates the indication grade of the memory area and the # indicates the immediate value.

- (a) A=#B;
- (b) A=B;
- (c) A=(B);
- (d) (A)=(B);



The sentence written with such design language is called pseudo code.

Detail Design can be realized through the Transformation
Process of Pseudo Code

Let's look at the progress of the actual design and coding by taking out part of the DLC of the Display Microcode (See the paragraphs (1) to (3) below). As the result of general design, (1) is obtained. The (2) is obtained by expressing the (1) in the pseudo code, specifying them to small movement and, rewriting (transforming) them to be able to perform coding. The (3) is obtained from the (2) by naming addresses, constants, etc. and then coding it. The logics for processing the exceptions missed during the transformation from (1) to (2) and from (2) to (3) might be added (the portion in (3), for example). Here we describe only 12 of 44 lines in (2) obtained from (1) and only 22 of 148 lines in (3).

(1) General Design Output

Read Message Available: When CAC receives a valid I-frame, this comp is returned. After receiving this completion. the received data is queued in Normal or Expedite Receive Queue according to the bit of the Header and validates RIP for further operation.

If the free buffer is no longer available, RIP validation is abandoned and RNR will be sent.

(2) Detail Design Output (12 of 44 lines)

```
IF PROG FLAG = READ MSG AVL THEN
DO;
  (RIP) = (PARMDCNT);          /* SET BCW COUNT TO RCVD DATA CNT
IF FM/¬FM BIT OF HEADER = ON THEN /* EXPEDITE
  IF EXP RCV Q LAST POINTER = INVALID THEN
  DO;
    /* EXP RCV Q HAS BEEN EMPTY
    EXP RCV Q POINTER = (RIP)-1;
    EXP RCV Q LAST POINTER = (RIP)-1;
    TO POINTER = (RIP)+3;      /* CLEAR RBCW TIC ADDR
    (TO POINTER) = X'FFFF';
  END;
ELSE
  :
```

(3) Source Code (22 of 148 lines, 27 of 149 bytes)

```
* IF PROG FLAG = READ MSG AVL THEN
  LDAA  PARMDF,D
  ANDA  #PFRDMSG
  BNE   DLCB4
*
  JMP   DLCIS          /* READ MSG AVL
*
DLCB4 EQU *
* DO;
*   (RIP) = (PARMDCNT);          /* SET BCW COUNT TO RCVD DATA CNT
  LDAA  PARMDCNT+1,D
  EORA  #X'02'
  BNE   DLCB5          NO
*
  DEC   DCBRIP+1      YES, VALIDATE RIP
  JMP   DLCJ8          IGNORE REQUEST
*
DLCB5 EQU *
  LDX   DCBRIP,D      NOW RIP POINTS TO COUNT
  EORA  #X'02'        RESUME ACCA
  STAA  #,X
*
  :
```

5005
IBM

Significance

- (1) Since the detail design stage is actualized as transformation process of the pseudo code, the same type of work as in coding or debugging using the display instead of a pencil and a rubber can be done. This enables speed up of overall works and the clear outputs which can be processed by the computer are issued. Thus, the maintenance is easier than in the case of flowchart.
- (2) The work at this stage is carried out by repeating the CMS work and the desk work described above. For these two, the latter is suited for the individual inspection after stepping down one stage, for example. However, how to use these two in proper rate depends on individual user's usability. From my experience, it seems that the former suits the local or simple work, and the latter suits the work which is global and requires deeper consideration.
- (3) It is strongly required such attitude that the documentation is a tool rather than the result of design.

- (4) There are some freedoms in design sequence, however, using these freedoms, basically, top-down design is naturally performed.
- (5) It can be an effective means for vendorization by stopping at the appropriate stage of the detail design (in an extreme case, end of general design=start of detail design) and offering vendor the pseudo codes obtained at that time.
- (6) The similar method can apply to the general design and the requirement specification stages. (Actually, at the time of this design, the functional specification of the display did not exist.)

Future Problem

(1) This successful example is such a limited case that the development scale is small (code size is about 4K bytes and only three person including a manager are involved) and the development phase is only from detail design phase. Therefore, its expandability becomes problems. The following are these examples:

1. In a large program, for each module we may succeed in the same way, but, how about the overall system?

How about documentaion for overall relation?

2. As documentation of the design result, what extent information can be transmitted effecively to the persons who are not working at the design stage all? Is it sufficient to couple with walk-through?

3. How much position is occupied in the overall develop-ment flow from the requirement specification stage to the general design? For example, in case of general design, to understand the overall image and the special condition, the block diagram, flow chart, and other related drawings are required. The pseudo codes for strict logics and the aforementioned may be used separately according to each problem arised and the circumstances of the user at a certain rate. Also, by building the mechanism which handles the implementation characteristics, it may open the way for the automatic design and coding. For the purpose, the formalizing of the requirement specification is necessary. However, at the stage of making the specification itself at most "semi-automatic" method will be available because it contains "reproducing of concepts".

- (2) Under individual conditions, which design language is suitable or best suited?

Acknowledgement

The author would like to express his special gratitude to Mr. Hiroyuki Ikeda for posing problem and his suggestions, to Mr. Junzo Kikuchi for providing the general design, and to Mr. Kunihiko Tejima for encouraging me to submit this paper.