AN ARABIC COMPUTER PROGRAMMING LANGUAGE GAREB

Dr. Mohammad Zeki Mohammad * & Sahar Abdul Aziz**

لغة برمجة عربية للحاسبة

(عريب)

الخلاصة :

نغرض [illegible] حاسبات [illegible] كثير من مواطنين العرب الذين لايعرفون اللغة الانكليزية من الضروري ابتداء تركيب لغة برمجة عربية للحاسبة تكون بسيطة للمبتدئين وسهلة التنفيذ على [illegible] كثير من الحاسبات الالكترونية • لذلك فقد تمت هذه المحاولة لانتاج لغة مشابهة للغة بيسك) ومكتوبة بلغة فورتران [illegible] سهولة نقلها من حاسبة لاخرى وقد سميت بلغة [illegible] لحروف [illegible] من (لغة برمجة عربية للحاسبة) [illegible] المحاولة على حاسبة [illegible] ذات ذاكرة تحوي 8000 كلمة •

لذلك فقد كانت هناك صعوبات [illegible] هذه اللغة [illegible] مكانية [illegible] جملة [illegible] تحتوي [illegible] متواضعة لانتاج لغة عربية [illegible] للحصول على حاسبة [illegible] عربية •

1- Introduction :

Printing in Arabic characters is available in many computer installations in the Arab world. However, programmers still resort to English in progrmming and also use it as an instruction media. In view of the necessity of the introduction of principles of computers to a large cross section of the Arab population whose English is poor, it would appear that a simple programming language in Arabic would be of great value.

There has been few attempts to produce compilers for arabic programming languages,e.g. Al-Khawarizmi (1) Laych (2) and others. Those attempts and the present one are only at their starting and the target of producing efficient compilers for simple arabic programming languages still need extensive research.

An attempt to introduce a simple language is presented by the authors. Two main conditions were kept in mind while starting this experiment.

Firstly the language should be simple and suitable for beginners. Secondly it should be possible to apply it on a large number of computer installations. This is why the suggested language is similar to BASIC for simplicity and coded in FORTRAN for portability.

This attempt was tried on one of the smallest IBM-1130 installations of 8K memory. Hence many limitations and difficulties were encountered. The results were quite encouraging and call for further application to other installations.

The package is called (GAREB) as the reverse of letters collected from the title (AraBic computEr progRamming languAGe) and its Arabic transliteration (عريب) which means strange or strange or stranger and is collected from the translation of the same title (لغة برمجة عربية للحاسبة).

2— Main Characteristics :

The set of programs forming the FORTRAN compiler to analyse Arabic programs, should be characterised by the following:

a. The compiler should take into account the fact that arabic programs should be written from right to left.

(b) Blanks in between characters and numerals should be permitted. The blanks are to be cancelled by the compiler.

* Director of computer centre, Mosul University

** Assistant Engineer in the same [illegible]

(c) The Arabic characters set should take a standardised form. The set of N.C.C. - Baghdad(8) was taken as basis. It is shown in Appendix I. The keyboard layout for the data entry is shown in Appendix II.

(d) A limited number of Arabic commands were tried because of shortage of core availability. Those are shown in Appendix III.

(e) Number of Arabic statements is limited mainly by the core size. For the 8K - IBM 1130, it was about 20 statements. In larger installation, mor sizable programs can obviously be executed using the same compiler.

3— Bakus Normal Form (BNF) :

To define the suggested language grammatically, the Bakus Normal Form is used. The set of grammar rules of BNF has a simple structure which can be used to define the syntax of the languages. The symbol : = used below means "is defined be", the operator ∫[ ] stands for " all the syntax items enclosed by brackets should be repeated " . the corner brackets are used to enclose Meta-Components. and the symbol | means " or " With these symbols, the syntax of the suggested language may be put as follows :

```
<READ STATEMENT>      : : = READ <VARIABLE NAME>∫[<VARIABLE NAME>]
<WRITE STATEMENT>     : : = WRITE <VARIABLE NAME>∫[<VARIABLE NAME>]
<ARITHMETIC STATEMENT> : : = LET <VARIABLE NAME> = <EXPRESSION>
<IF STATEMENT>        : : = IF <EXPRESSION> <GO TO STATEMENT>
<GO TO STATEMENT>     : : = GO TO <LABEL>
<VARIABLE NAME>       : : = <SYMBOL><DIGIT>|<SYMBOL> <SYMBOL>
<EXPRESSION>          : : = <OPERAND>∫[<OPERATOR> <OPERAND>]
<OPERAND>             : : = <SIGN> <VARIABLE NAME> |<CONSTANT>|<SUBEXPRESION>
<SUBEXPRESSION>       : : = (<EXPRESSION>)
<LABEL>               : : = <INTEGER>
<DATA STATEMENT>      : : = DATA <CONSTANT>∫[<CONSTANT>]
<CONSTANT>            : : = <SIGN> <DECIMAL> | <SIGN> <INTEGER>
<DECIMAL>             : : = <INTEGER> <FRACTION> | <FRACTION>
<FRACTION>            : : = . <INTEGER>
<INTEGER>             : : = <DIGIT>∫[<DIGIT>]
<DIGIT>               : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<SYMBOL>              : : = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
                            Q | R | S | T | U | V | W | X | Y | Z
<OPERATOR>            : : = + | - | * | / | ↑
<SIGN>                : : = + | - | ε null
```

APPENDIX I

N.C.C. - Baghdad

ARABIC CHARACTER SET

LATIN STD CHARACTERS	CARD CODE	EBCDIC VALUE	ASCII VALUE	ARABIC REPLACEMENT
^	11-8-7	5F	5E	ؤ
_	0-8-5	6D	5F	ى
`	8-1	79	60	ا
a	12-0-1	81	61	ب
b	12-0-2	82	62	ت
c	12-0-3	83	63	ث
d	12-0-4	84	64	ج
e	12-0-5	85	65	ح
f	12-0-6	86	66	خ
g	12-0-7	87	67	د
h	12-0-8	88	68	ذ
i	12-0-9	89	69	ر
j	12-11-1	91	6A	ز
k	12-11-2	92	6B	س
l	12-11-3	93	6C	ش
m	12-11-4	94	6D	ص
n	12-11-5	95	6E	ض
o	12-11-6	96	6F	ط
p	12-11-7	97	70	ظ
q	12-11-8	98	71	ع
r	12-11-9	99	72	غ
s	11-0-2	A2	73	ف
t	11-0-3	A3	74	ق
u	11-0-4	A4	75	ك
v	11-0-5	A5	76	[illegible]
w	11-0-6	A6	77	[illegible]
x	11-0-7	A7	78	ل
y	11-0-8	A8	79	و
z	11-0-9	A9	7A	ط
{	12-0	C0	7B	[illegible]
}	11-0	D0	7D	—

4— Procedure of Operation :

The FORTRAN compiler performs the following operations.

4.1 Entering all the Arabic character set in Al-Format. As there are no Arabic symbols on the card punch keyboard, their latin equivalent of Appendix I was introduced. The 47 elements comprising of alphabetic numeric and operators etc were stored in LT.

4.2 Entering the Arabic commands given in Appendix III to be stored in LK.

4.3 Entering the Arabic program which is punched on data cards of 80 columns. It is read as Al-Format and stored in LR. The program is listed as punched on the data cards.

the array U since the convention of formation of numbers out of digits in Arabic is the same as English. The statements at this stage can be listed in their correct form on the printer

4.4.4. The type of the statement using the subroutine TYPE is found by giving a code for each type according to the NTYP given in Appendix III. This subroutine also computes the numerical values in I-Format for the statement numbers, given in A-Format using the FUNCTION GET.

4.5 LEXICAL ANALYSIS :(7) The most basic phase of any translation is that in which the input program is subdivided into its elementray constituents : identifiers, operators, symbols, numbers, ets. This phase is termed lexical analysis, and the basic program units which result from lexical analysis are termed lexical items. Typically the lexical analyzer LCLAS given in Appendix IV is the input routine for the translator, reading successive lines of input program, breaking them down into individual lexicel items, and feeding these lexical items to the later stages of the translator to be used in the higher level of analysis. Thus the lexical analyzer is a translator whose input is the string of symbols representing the source program and whose output is a stream of lexical items. This output forms the input to the syntactic analyzer.

4.6 SYNTACTIC ANALYSIS (PARSING) : The second stage in translation after the lexical analysis. Here the larger program structures are identified : statements, expressions, etc. using the lexical items produced by the lexical analyzer (LCLAS). Syntactic analysis usually allernates with sementic analysis (4.7). First the syntactic analyzer identifies a sequence of lexical items forming a syntactic unit such as an expression or statement using subroutine LHKVR. A semantic analyzer is then called to process this unit. The syntactic analyzer enters in a tack the various elements of the syntactic unit found, and these are retrieved and processed by the semantic analyzer.

4.7 SEMANTIC ANALYSIS : Semantic analysis is perhaps the central phase of translation ... the syntactic structures recognized by the syntactic analyzer are processed and the structure of the executable object code begins to take shape.

Semantic analysis is thus the bridge between the analysis and synthesis parts of translation. A number of other important functions also occur in this stage, including symbol table maintenance, most error detection. The output from this stage is some internal form of the final executable program.

The semantic analyzer is ordinarily split into a set of smaller semantic analyzers given in Appendix III as syntax subroutines, each of which

4.7.1 Handles one particular type of program construct. For example, erithmetic expressions might be handled by one analyzer, GO TO statements by another, and READ statements by another.

4.7.2 The appropriate semantic analyzer is called by the syntactic analyzer whenever it has recognized a syntactic unit to be processed

4.7.3 The semantic analyzer interact among themselves through information stored in various data structures, particularly in the central symbol table

4.7.4 These subroutines contain the necessary flags for

Appendix IV

Coding for Parsing

Code (LCLAS)	Object
1	Numeric Characters
2	Alphabetic Characters
3	Operators
4	Decimal point
5	Equal sign
6	Comma
7	Prefix (minus or plus)
8	Left parenthesis
9	Right parenthesis

syntax errors giving a code for each type of error with reference to the line number where it occured.

4.7.5 If the statement is correct syntactically then the operation of Tables construction (extraction operation) is performed. The following tables are to be constructed

(a) Table NO containing the numbers in the same sequence as they come in the program

(b) Table NTYP containing the code numbers for the statements.

(c) Table LT containing the nature of the contents of each statement.

(d) Table MC containing the limit for each statement

By the construction of these tables the parsing phase is finished Hence the program is ready for interpretation. The above operation should be repeated for all the Arabic program cards

4.8 CODE GENERATION This process involves formating the output properly using the information contained in the internal program representation in this stage the SYMBOL TABLE which contains information about each variable will be built The variables are defined in the SYMBOL TABLE before its usage in the program.

4.8.1 Codes are generated by the subroutine VRTBL which tables the vriables from the read statements in the program and loads them in the array VAR Simultaneously the corresponding value of that variable is taken from the data statement using FUNCTION C... and converted to executable numbers This value is loaded in a table called VAL Before loading any variable, it should be checked whether it already exists in the table or not. The table then contains the variables in all the read statements and their values

A pointer named KJ gives the limit of the variables contained in the read statement from the table. This table is completed by adding the variables coming from the table left-hand side of the assignment statement. This is done using ASTBL which adds the variables to VAR keeping corresponding values in VAL blank till the mathematical operations are performed then their values are substituted. The NTYP table is to be scanned for assignment statement. If any non-executable number are found within the statement then they are converted into executable statements using FUNCTION CET. This is performed using the subroutine CNVRT.

4.8.2 Subroutine SOLVE is called to perform mathematical operations within assignment statemnets. The assignment operation produces a side effect-achange in the value of a variable or data structure element. The side effects produced by one statement affects the input of the next statement in sequence(?

For this reason, the NTYP table is to be scanned for assignment statements in the same order as the source program The assignment statement is then moved to a new table called MATH which is used as a working table to do mathematical operations after substituting the value of the variables form the symbol table. This will save the variables in the assianment statements when it is used for more than one iteration if a DO, or GOTO statements (i e) control statements are used withim the program. Now, MATH consists of operands in executable form separated by operators and ready for code generation using RE-ORDERING technique which searches for the operator in this order (↑, /, *, +, –) When the operator is found, it become easier to find the adjecent 2 - operands (the operand preceding and the operand following the operator) and do the mathematical operation according to the operator found(4) Then it assigns to the result a variable name within the table and reorder the table and continues in the same fashion for all the operators in the statement The final result for the assignment statement is then moved to the SYMBOL TABLE.

5. Discussions and Conclusions .

5.1 Limitations

Appandix V shows a solved example using the compiler described in this paper There were few limitations on writing such a program such as the following

5.1.1 Maximum number of statements is 20 in the whole program

5.1.2 Maximum number of statements of each command type is 5

5.1.3 Variables may consist of one or two characters .

5.1.4 The FORTRAN programming available on IBM 1130 makes it difficult to convert variables from A-Format to A2, A3 ... ect If such a facility is available on any system, then use of polish form