INTRODUCTION
The ARC (Argonne Reactor Computation) System has been developed to facilitate the studies required for fast-reactor design. The areas of physics, safety, fuel utilization and core-design had initial priority. Other general engineering capabilities will be added later.

Reactors have traditionally been designed by means of stand-alone codes. However, it is rarely the case that a single code will suffice for an entire calculation much less for an entire reactor design. Usually, the solution of one problem becomes the input to another problem, but not without the intervention of conversion programs. This procedure is followed until the complete solution is effected.

It was inevitable that the reactor industry would sponsor the development of systems to automate the design process. The first effort in this direction was undertaken by the Knolls Atomic Power Laboratory with the NOVA project in 1964. The ARC project at Argonne National Laboratory was begun in 1965. Still later, the JOSHUA project was begun in 1968 at the Savannah River Laboratory.

The ARC system, when completed, will encompass the field of reactor computations and automate what was once a series of interrelated programs. A glossary has been constructed that defines and formats all of the types of data that can be expected for reactor calculations. In addition, a library of programs has been built that will perform the mathematical and data handling algorithms that occur in reactor computations.

A “system” capability has been provided that allows the library and data to be used in a flexible, coordinated manner. Reactor calculations can be accomplished by designing a director program that executes modules from the library in the proper sequence to operate on and produce data.

Although such a system can be specified independently of a choice of computer, the implementation is not computer independent; it depends upon a choice of machine and the manufacturer’s software. ARC is currently running on an IBM System/360 50-75 combination using ASP. The system runs as an ordinary job under OS/360 and it will run on any configuration which uses OS/360. Practically speaking, ARC needs a large amount of core and auxiliary storage, and a fast CPU. These restrictions are caused by the nature of reactor calculations rather than the nature of ARC; reactor calculations involve a lot of computation and require extensive data manipulation and storage capabilities.

Design features
Original specifications

The original specifications for ARC required the following features:

1. Modular approach. Any major computational program consists of a collection of computational units, certain of which are common to many computations. A modular approach facilitates the construction of such programs. In addition, the modular approach provides a maximum amount of core for each module provided the system storage requirements remain small.

2. Common data base or data pool. In a modular approach the data associated with each module must be potentially available to any other module in the system. This implies the existence of a
data pool which is responsible for intermodule data management.

3. All coding in FORTRAN IV. If all coding is done in high-level languages, then program interchange is greatly facilitated.

4. Use of manufacturer's software whenever possible.

5. Ease of use. This is probably part of the specifications for all systems and is an area where much time can be spent even after a system becomes operational.

Original implementation

Surprisingly, a simple initial approach satisfied all of the criteria except 5. This approach was a huge overlay program with system functions in the root segment, paths in region one and computational modules in region two.

The overlay system shown in Figure 1 was in production before it collapsed of its own weight. Two flaws dictated the approach described in this paper:

1. Overlays allow a limited number of external symbols, and duplicate names are forbidden.
2. The system had to be reconstructed whenever any change occurred anywhere in the system.

A small amount of assembly language coding turned the overlay system into a dynamic loading system. The computational modules of the overlay were assimilated into the present system with few changes.

The dynamic system

The main components of ARC

1. A library of programs (OS/360 load modules).
2. ARC system subroutines that are useful to most modules of the library.
3. A data set glossary that defines the data sets.

In addition, the ARC system is predicated on the use of OS/360 with data in the form of OS/360 data sets.

Modules in ARC

ARC contains three kinds of modules, computational, control, and system.

1. A computational module is defined in terms of its input, output, and the algorithms that manipulate the input to produce the output. Input to computational modules is in the form of data sets and parameters lists; output is in the form of data sets and printed output. All computational modules are written in FORTRAN IV and are loaded and executed by control modules.

2. A control module is a FORTRAN IV program that can be executed as the main program of ARC. A control module or path is directive in nature; it causes computational modules to be executed in the proper sequence to effect a series of calculations. A path performs the following functions:

   a. Initialization of core to provide an environment for the computational modules.
      i. Permanently loads resident system modules.
      ii. Initializes tables.
      iii. Processes card input for later use.
   b. Directs the computation.
   c. Calls for subsets of the input data as they are required.

Paths are classified as standard if they exist in the library at the start of a run or nonstandard if they are compiled at run-time. Any path is allowed to use another path as it would any computational module. This allows a new calculation to use an existing calculation if it performs a useful subcalculation.

In order to initialize the system, a path must contain two initialized arrays. The first array contains the names of all of the data-sets used by the path and by any module called by the path. The placement of the name in the array determines the data set reference number associated
with the name. The second array lists the names of blocks of input data that are required to perform the calculation. Input data is divided into blocks with cards of the form

\[
\text{BLOCK} = \text{NAME}_n
\]

where \( \text{NAME}_n \) is an 8-character identifier. When a path performs

\[
\text{CALL DATA(NAME, N)},
\]

a table that describes the input data set is searched for the occurrence of the block NAME. If a block by that name that has not been processed exists, it is formed into data sets. Block names can be duplicated, and repeated calls of the same name process the next block by that name; when all blocks with that name have been processed, that fact is reflected in \( N \). This facility is useful in repeated runs of a path. The absence of a block can terminate the run or cause another logical decision to be made.

3. **ARC system modules** perform functions that are required by paths and computational modules. They are of two types:

   a. Resident. These are permanently loaded into core and executed by the first path to gain control. Resident system modules perform the functions of system initialization, input data processing, data management, and internal table maintenance.

   b. Transient. These modules are executed by certain of the resident system modules. Their primary function is to process input data. One module is used during initialization to convert input data into a data set and to describe the properties of that data set in tables held in core. Another module is used when a path requires a subset of the input data.

**System subroutines**

The second main component of ARC is a collection of system subroutines which form an essential part of most of the modules in the system. These subroutines allow the FORTRAN program to invoke certain of the OS/360 macros, provide an interface to the resident I/O module, and communicate with the system. A group of system subroutines, collectively called POINTR, provides a variable dimensioning capability and is included in the modules that require this feature.

The system subroutines LINK and LOAD invoke the MACROS with the same names.

1. **LINK** is an assembly-language subprogram callable from a FORTRAN subprogram via the statement

\[
\text{CALL LINK (name, additional parameters)}
\]

where “name” is either an 8-character alphanumeric quantity or a “REAL*8” variable containing such a quantity. The quantity must be the name of an ARC module. The subprogram LINK causes the OS/360 “LINK” macro-instruction to be invoked and results in control being transferred to the entry point of the module “name.” If “name” is not currently in core, the “LINK” macro loads “name” into unoccupied core before giving it control. Any additional parameters in the above call to LINK are passed to “name.” When control exits from “name,” it returns to the calling subprogram at the statement immediately following the call to LINK.

LINK may be used to pass control from one program module to another in exactly the same fashion as the FORTRAN “CALL” statement is used to pass control from one subprogram to another within the same module.

2. **LOAD** is an assembly-language subprogram callable from a FORTRAN subprogram via the statement

\[
\text{CALL LOAD(name)}
\]

where single parameter is either an 8-character alphanumeric quantity or a “REAL*8” variable containing such a quantity. The quantity must be the name of an ARC program module. The subprogram LOAD causes the OS/360 “LOAD” macro-instruction to be invoked and results in the loading of the module “name” into unoccupied core. If the module “name” is already resident in core, then either (i) no action is taken (if the module was declared reusable), or (ii) another copy of “name” is loaded (if the module is not reusable).

3. Modules locate data by using the system subroutine DSRN. DSRN is called from a FORTRAN subprogram via the statement

\[
\text{CALL DSRN (name, N1, N2)}
\]

where “name” and N2 are input parameters, and N1 is an output parameter.

“Name” must be either an 8-character alphanumeric quantity or a “REAL*8” variable containing such a quantity; that is, one of the glossary names in the array DSNAME as
initialized in the path control module of the path being executed.
The function of DSRN is to return to the calling program in N1 a data set reference number corresponding to the data module “name.” In this way data set reference numbers are global to the system instead of local to the modules. The subroutine DSRN simply LINKS to a resident system module that performs the table search.

4. Data management within fast memory is accomplished by the subroutines of POINTR. Each subroutine of POINTR performs a specific data management function and as such provides a “pseudo-instruction” in a set of data management instructions. Thus, the FORTRAN language has been extended to allow data packing, storage allocation, and storage freeing in a manner similar to that available in the PL/I language.

Control of data in fast memory remains under the control of the module programmer, who provides POINTR with a large “container” array into which data arrays will be placed. POINTR contains tables which keep track of arrays being stored in the container, thus relieving the programmer of this burdensome chore. Recorded in the tables are the array name (usually identical with the FORTRAN variable name), length, starting location, and FORTRAN variable type. Subsequent retrieval of arrays is by array name.

5. Management of external data within a module is through the standard FORTRAN I/O routines and the data set is the organizational unit. This obvious approach simplified the initial programming but created a rather rigid data structure. A large amount of effort went into the construction of a glossary that describes all of the data sets that have been written. The glossary is open-ended and at all times contains the definitions and description of all data sets which are used or created in ARC. In addition, the first record of most data sets lists the contents and important parameters.

This discussion of the components of ARC shows that OS/360 is a foundation of the system. Through the implementation, a guiding philosophy has been “peaceful coexistence with OS/360.”
A sample program

The following statements show the coding that is necessary to compile and execute a nonstandard path (PATH1). Execution of PATH1 causes the events depicted in Figure 2 to take place. Figure 2 is a dynamic picture of core with time increasing from left to right. ARC system modules are marked with an asterisk.

The following statements show the coding that is necessary to compile and execute a nonstandard path (PATH1).

```
//NSP JOB
//JOBLIB DD DSNAME=MODULES,DISP=(OLD,PASS)
//STEP1 EXEC FTHCLG
//FTH.SYSIN DD *
C DD
C PATH1
REAL*B DA(240)://'A.A','A.B','XYZ','S'/'
REAL*B DS(10)://'BLOCK1','BLOCK2'/'
C CALL SYSTEM(DA)
CALL DATA(DS(1))
CALL LINK('MOD1 ')
CALL LINK('MOD2 ')
CALL LINK('PATH2 ')
C
RETURN
END
/*
//EDT.LIB DD DSNAME=MODULES,DISP=(OLD,PASS)
//EDT.SYSIN DD *
INCLUDE LIB(SYSTEM)
/*
//90.FT09FOO1 DD DISP=NEW,UNIT=DISK,SPACE=(TRK,(10,10))
//90.FT11FOO1 DD DSNAME=A,A ...........
//90.FT12FOO1 DD DSNAME=A,B ...........
//90.FT13FOO1 DD DSNAME=XYZ ...........
//90.SYSIN DD *
BLOCK=BLOCK1
DATASET=A,A
{Data for A,A}
DATASET=A,B
{Data for A,B}
BLOCK=BLOCK2
MODIFY=A,A
{Cards to modify data set A,A}
/*
```

The FORTRAN IV program PATH1 is compiled during time period 1 (TP1). At TP2 the path is in core and the I/O initialization of FORTRAN causes the ARC I/O module to be LOADed into core. TP4 and TP5 are related to the system initialization and make use of STMT1, STMT2, and STMT3 of PATH1. STMT3 actually causes system initialization utilizing the information in STMT1. Three data sets are expected to be used (A,A, A,B, XYZ) and input data are divided into two blocks (BLOCK1 and BLOCK2).
STMT4 corresponds to TP6; it causes BLOCK1 to be processed into data sets A.A and A.B. STMT7 is responsible for TP9, TP10, TP11. PATH2 processes BLOCK2 into a modification of data set A.A. In STMT12, a module named SYSTEM is being combined with PATH1. SYSTEM contains all of the ARC system subroutines that a path requires.

This simple example shows that the internal workings of the system are simple and flexible. Since a path is a FORTRAN program, complex logical structures pose no problems; they are composed of familiar statements. The computational modules are also composed of FORTRAN statements and the only restriction is that a computational module never tries to initialize the system.

Current capability

Our program library presently contains 40 modules. Twenty-two of these are computational modules, twelve are control modules and six are system modules. In addition there are nine system subroutines that are part of each computational and control module. These subroutines interface with the system modules.

The average computational module contains about 2500 statements. About half of the computational modules are overlay programs and subroutine standardization is utilized as much as possible.

Critique and future plans

Problem areas

ARC is an ambitious project pursued by a small number of people. As such the project has been subjected to compromises dictated by limitations in manpower and time. Early in the project a decision was made to expend the greatest effort on the computational aspects of the system and to rely as much as possible on OS/360. Consequently some of the desirable features of the originally specified system were deferred or compromised. A brief criticism of the present system will be enlightening.

Any large, modular scientific programming system is dominated by the problems of data management. These problems are in two areas:

1. management of program modules, i.e., the ability to link computational modules in a complex manner;
2. management of computed data, the ability to store and retrieve data.

Program management

While adequate linkage facilities are available to the module writer, job executions are not convenient for the user. The Data Definition statements of OS/360 are the external data language for this system. The quantity and complexity of these DD statements cause many errors to be made. Thus, the reactor designer may have little difficulty in designing and writing a new path, but have great difficulty making it work. In practice, the programmer must be quite adept to some of the most sophisticated uses of OS/360.

Data management

The method of computed data management within the ARC system is also not satisfactory for a third-generation system, both from the viewpoint of the reactor designer and the program developer. In any large system there is always an inherent conflict between the structure of the language of the user and the internal structure of data necessary for efficient operation of the system. In ARC we have compromised. The Data Definition statement of OS/360 is not oriented to the reactor designer. The rigidly structured external files used in ARC cause two classes of problems.

1. They are not conducive to efficient manipulation other than for the specific use that dictated their initial structure. Future arbitrary use and manipulations of these files or, more importantly, arbitrary combinations of parts of these files must necessarily be time-consuming. If, for example, subsequent references require that the file structure should be inverted, the ordering is in conflict for efficient operation; etc.
2. Modularity of program development is greatly hampered by the rigidly structured files because redefinition of the file structure in files produced by any module must be reflected in all modules referencing that file. This modularity would be enhanced if data references were instead oriented to FORTRAN variables. Further complexity is introduced since the module developer must concern himself not only with data of interest to his module but all quantities in each external file necessary for him to reference.

Proposals for improvement

These problems have generated specifications for two subsystems:

1. A generalized data management package that would allow modules to reference all data by FORTRAN variable names irrespective of core storage requirements. This package would have taken on the responsibility of all core manage-
ment and external file manipulation by means of high frequency, priority usage algorithms. Thus it would be transparent to the module writer if variable names that were referenced in the module required external file manipulation or not; i.e., the system would provide him with an effective infinite memory. This proposal addressed itself to the total problem of management of computed data as well as enhancing modularity of program development. This ambitious project was not undertaken due to lack of resources.

2. A system that addresses itself to the problems of program management and external file management. This system is comprised of a program maintenance package, a datapool maintenance package, and a job control language translator. This too is an ambitious project that would have general usage outside of the ARC system.

Current plans

Our current plans are evolutionary rather than revolutionary. Three aspects must be considered:

1. Internal algorithms.
2. Data management.
3. External language.

The internal algorithms are constantly reviewed and are improved wherever possible. Improvements in algorithms should not negate work that has been done in other areas.

Data management will be augmented; most of the data sets that have been defined will remain, but alternate facilities will be provided. In particular, we expect to construct a variable-oriented data pool. When this is completed, some of the data sets will be separated into many members that can be accessed directly. In particular, certain important variables will be referenced by name throughout the system.

An external language is required to ease the burden of running a computation that is in production status. The deck of Job Control Language that is required to run a job is large. While it is true that a user can accomplish all of the external data set manipulations that are required, he must specify his needs in JCL, and this is clumsy. Preliminary plans have been made for a simplified language that will produce the JCL required to run any job that is in production status.

This system, while designed for the reactor industry, has generality. With a suitably defined glossary and computational modules, it can be the basis for a computational system for other areas.

REFERENCES

1 B J TOPPEL
Reactor computation development
Reactor Physics Division Annual Report July 1 1964 to June 30 1965 ANL-7110 p 339
2 E D REILLY JR W H TURNER
The automation of reactor design calculations at KAPL
3 B J TOPPEL
The Argonne reactor computation system
Reactor Physics Annual Report July 1 1965 to June 30 1966 ANL-7210 p 349
4 C N KELBER G JENSEN I JUST B J TOPPEL
The Argonne reactor computation system ARC
Proc of the International Conference on the Utilization of Research Reactors and Reactor Mathematics and Computation p 1428
5 B J TOPPEL
The Argonne reactor computation (ARC) system
6 B J TOPPEL
The Argonne reactor computation system ARC
ANL-7332 1968
7 L C JUST S D SPARCK
The ARC system
AMD Internal Technical Memorandum No 157 1968 unpublished
8 J E SUICH J C JENSEN H C HONECK
JOSHUA—An operating system for reactor physics computations

From the collection of the Computer History Museum (www.computerhistory.org)