COMPUTATIONAL STRATEGY FOR THE SOLUTION OF LARGE STRAIN NONLINEAR PROBLEMS USING THE WILKINS EXPLICIT FINITE-DIFFERENCE APPROACH

R. Hofmann

Science Applications, Incorporated
2450 Washington Avenue, Suite 120
San Leandro, California 94577

SUMMARY

The STEALTH code system, which solves large strain, nonlinear continuum mechanics problems, has been rigorously structured in both overall design and programming standards. The design is based on the "theoretical elements of analysis" while the programming standards attempt to establish a parallelism between physical theory, programing structure and documentation. These features have made it easy to maintain, modify and transport the codes. It has also guaranteed users a high level of quality control and quality assurance.

INTRODUCTION

A computer code system called "STEALTH" (ref. 1)*, has been developed for the Electric Power Research Institute (EPRI) for the primary purpose of solving nonlinear, static, quasi-static and transient problems involving both fluids and solids. The numerical technology for this computer program is based on the developments of Wilkins (ref. 2) and Herrmann (ref. 3). Although this technology was originally developed for large deformation, fast-transient defense-oriented applications (figure 1), it has been adapted to be quite useful for studying thermal-hydraulic mechanical transients (figure 2), nuclear waste isolation geologic burial stability (figure 3) and a variety of structure-medium interaction (SMI) problems (figures 4 and 5). The design and development of general-purpose

---

*"Solids and Thermal hydraulics codes for EPRI Adapated from Lagrange TOODY and HEMP", developed for Electric Power Research Institute by Science Applications, Inc., under contract RP307.

STEALTH involved extensive planning in order to make it adaptive enough to handle this wide variety of nonlinear problems. This paper describes the strategy that was (and still is) used.

ARCHITECTURAL OVERVIEW

The overall structure of the STEALTH nonlinear code system has been built around a particular view of the physical equations being solved. This view is based on the "theoretical elements of analysis of a physical system" which are summarized in Table 1. The theoretical elements of analysis are a convenient conceptulization for solid and fluid mechanics problems. It separates physics laws, material response characteristics, geometric aspects (e.g., boundary conditions) and initial conditions. These distinct categories are not only convenient theoretical groupings, but are also useful programming and documentation entities.

The STEALTH architecture based on this view has stood the test of time for over five years and continues to be quite flexible and adaptable to new problems and more complex situations. Among the many adaptive features are (1) the ability to couple other computer programs, (2) a standard procedure for externally developed constitutive models, (3) a modular topdown architecture with a FORTRAN syntax that makes developing and changing subroutines easy, and (4) a general-purpose, special-purpose version arrangement that guarantees good quality assurance. Finally, it has been possible to add new capabilities that were not specifically anticipated when STEALTH was originally designed.

STEALTHs 1D, 2D, and 3D are based on a modular architecture in which many subroutines and COMMON blocks in each code are identical in every detail. The top-down design that was implemented requires each code to have the same calling sequence at its highest levels. Subroutines and COMMON blocks which must be different are found at the lowest (innerrmost) levels of STEALTH. In between, there are subroutines that have identical names, functions, and structure, but different specific programming.

The actual FORTRAN programming utilizes a subset of FORTRAN that is common to IBM, Univac, and CDC computers. The use of these FORTRAN statements is further restricted by format conventions that produce very structured programming. In addition, FORTRAN variable names are formed by combining three-character roots with one- and two-character prefixes and suffixes.

The STEALTH codes have been designed to be most efficient for the occasional user. The standard version combines extensive checking logic which checks and rechecks a user's input and checks and rechecks the status of the calculation as it proceeds. The codes also provide many standard models for materials, boundary conditions, etc.

Computer memory requirements range from 135 000 to 155 000 words of octal storage in CDC 7600. This size has been achieved by overlaying the GENERATOR. Further reduction of code size can be achieved by either overlaying the GENERATOR some more, overlaying the PROCESSOR, or by reducing the size of certain COMMON blocks. Reducing the size of COMMON blocks usually results in a reduction in the number of grid points that can be computed.

Other tailorings of the codes can be made to suit specific computing environments and/or problems. For example, it is simple to put "hardwired" material models into the code to improve code speed. It is also possible to remove the trace and debug options, again improving speed. For short production runs where generation is a larger proportion of the run time, it is possible to write a pregenerator to reduce GENERATOR costs. Finally, special-purpose versions of STEALTH can be created in order to improve efficiency. For example, a hydrodynamic-only version of STEALTH runs 20% faster than the standard version for the same fluids problem.

The FORTRAN coding conventions and the structural modularity make STEALTHs 1D, 2D, and 3D portable and device-independent. Word size and memory storage limitations are determined from the requirements of an actual calculation. For most calculations, it is desirable to use a machine which has a word size greater than 48 bits and memory of at least 30 000 decimal words. However, it is possible to perform STEALTH simulations at a word size of 32 bits and a memory of 20 000 decimal words. The STEALTH code system is made up of more than 100 000 FORTRAN cards.

PROGRAMMING STRUCTURE

The development of a user-oriented, well documented, Wilkins explicit finite-difference computer code is based on the premise that programming structure, input/output, and documentation should be formulated from physical rather than mathematical (or numerical) concepts. Theory, code structure, and documentation are fundamental categories in the discussion of user orientation. Elements of these categories (Table 2) should be as similar in vocabulary and notation as possible. Using the theoretical elements of analysis as the basis for this design, automatically links the physical theory and programming structure in a way in which program development is easily achieved. In the discussion that follows, a standard view of program structure has been adapted to these concepts.

The STEALTH computer programs do numerical simulations as opposed to numerical evaluations. A simulation is carried out through execution of three separate "phase groups". Appropriate names for these phase groups are GENERATOR, PROCESSOR, and OUTPUT ANALYZER. (Analogous processing concepts exist for computer systems. They are: compiler/loader, central

processing unit (CPU), and output devices, respectively.) The conceptual functions of each phase group are summarized below.

- The GENERATOR accepts detailed data (from cards or keyboard) as input for many different types of computer calculations. All input data are checked as thoroughly as possible. If no serious errors are detected, an input file for the appropriate PROCESSOR phase group is prepared.

- The PROCESSOR accepts preresolved (link-edited) data from the GENERATOR as a complete specification for a calculation in order to perform a specific physics calculation. During the calculation, output data are prepared to be input for both the OUTPUT ANALYZER and the GENERATOR. These data take the form of archive files. The file for the GENERATOR is called the restart file, while the data for the OUTPUT ANALYZER are known simply as archive data. The PROCESSOR is analogous to a CPU. Its primary purpose is to compute (crunch numbers) and direct data to output devices.

- The OUTPUT ANALYZER accepts data from the GENERATOR or PROCESSOR phase groups in archive format. It performs analysis functions such as plotting, special printing, data reduction, etc. Output data from the OUTPUT ANALYZER are presented either in hard copy form or as input files for other OUTPUT ANALYZERS. It is exactly analogous to hard copy output functions of a hardware printer, plotter, or other output device.

Figure 6 is a schematic display of the interaction between phase groups.

The GENERATOR phase group is a combination compiler/loader. For example, input to the compiler function of the GENERATOR is the code input for a particular problem. The loader (link-editing) function of the GENERATOR performs the task of resolving several types of input into a single file to be read by an appropriate PROCESSOR. The GENERATOR is capable of setting up (loading) a variety of problems from a spectrum of input modes. There are two GENERATOR input modes, standard and nonstandard.

- Standard Record Format (start or restart)

  cards
  keyboard

- Nonstandard Input (start only)

  library file

The PROCESSOR phase group is analogous to the CPU of a computer. Its purpose is to compute physics from appropriate algorithms. Input data for the PROCESSOR are prepared by the GENERATOR. Output data are in the form of an archive file. The archive file contains a complete summary of all the results of the physics calculations performed. The format of the file is a self-contained, easy-to-read format and is designed to be used by other programs as input (e.g., programs in the OUTPUT ANALYZER phase group can read the archive file as input). An abbreviated form of the archive file, known as the restart file, is created as an input file for the GENERATOR. If a calculation must be restated, it is more convenient to use the abbreviated file than a complete archive file.

The OUTPUT ANALYZER phase group is composed of many different stand-alone computer programs. Among these are plotting programs, Fourier analyzers, data reduction codes, etc. Output analysis may be performed on data from both the GENERATOR and PROCESSOR phase groups. All data are transmitted in archive format but only the PROCESSOR creates a permanent archive file. (The OUTPUT ANALYZER can be used to make a reduced archive file, if required.)

Output from the OUTPUT ANALYZER phase group is usually in the form of hard copy (that is, printed pages, plots, etc.). Files that are produced as output are usually in archive format also so that they can be used as input for other data analyzing functions. These files are not intended to be used as input to the GENERATOR or the PROCESSOR, although it is conceivable that they could be used this way.

Phases are the logical subdivisions of a phase group. They are groups of subroutines which perform a particular logical "macrofunction" which preserves the simplicity of physical concepts. In contrast, subroutines perform "microfunctions" and are defined by a specific functional task such as reading, checking, calculating, etc., or a well defined combination of these tasks.

A phase group can be divided into as many phases as necessary. One special phase known as the Utility phase is part of every phase group. It contains subroutines which are used by more than one phase and which fall into one of the following categories:

(1) System or Machine Dependent
(2) Input Related
(3) Output Related
(4) Enter/Exit
(5) Error
(6) Arithmetic
(7) Miscellaneous

All phases are chosen from logical or conceptual considerations dictated by the tasks to be performed by a particular phase group.

Phases in the OUTPUT ANALYZER cannot be defined a priori. This phase group may require different phase structure for different types of problems and different types of analyses, respectively. However, the GENERATOR and PROCESSOR phases are amenable to a general design concept based on the physical notions associated with the partial differential equations being solved.

The fundamental ideas behind the design of the phase structure for the GENERATOR and PROCESSOR phase groups of STEALTH come directly from the theoretical elements of analysis of a physical system. That is, logically, STEALTH may be viewed in terms of the following distinct subdivisions:

- Conservation Equations

  mass
  momentum
  energy

- Boundary Conditions

  geometric constraints
  boundary values

- Initial Conditions

  intensive
  extensive

- Constitutive Equations

  mechanical
  thermal

Designing the GENERATOR and PROCESSOR for STEALTH from these four categories is relatively straightforward. The conservation equations (equations of change) are the equations to be solved; they are the kernel of the PROCESSOR. They describe the response or motion of a physical system. The initial and boundary values and the constitutive equations supply the conditions or constraints for solution. The computational network is formed from the geometric constraints and the time-dependence specification.

The STEALTH GENERATOR phase group is broken down into nine phases which are further divided into two groups. The first group contains two non-optional phases which must be executed prior to the execution of other GENERATOR phases and the Utility phase. The second group is composed of six

optional phases which are used selectively to satisfy specific processing requirements of a particular PROCESSOR. While the two nonoptional phases in the former group must be executed in a particular order, the phases in the latter group may be executed in any order.

The two phases in the nonoptional group are called CMNGEN and PRBGEN. CMNGEN initializes all common blocks and PRBGEN provides data for a GENERATOR scheduler. The GENERATOR scheduler is a subroutine which determines which of the latter group's optional phases is necessary for a particular PROCESSOR.

The functions of the optional phases in the GENERATOR phase group are (1) material model definition, MATGEN, (2) mesh or grid-point generation, GPTGEN, (3) zone interior initialization, ZONGEN, (4) boundary value specification, BDYGEN, (5) time control, TIMGEN, and (6) edit specification, EDTGEN. Figure 7 is a flow chart of the phases in the GENERATOR phase group. Table 3 shows the correspondence between the six optional GENERATOR phases and the logical elements of design.

Within each GENERATOR phase the subroutine calling structure (logic) is similar. Each phase contains a phase scheduler subroutine which calls all the "mainline" subroutines. The scheduler name is _ _ _GEN, where _ _ _ is the phase name.

The mainline subroutines are an input processing subroutine, _ _ _INP; a subroutine that checks input data, _ _ _CHK; a subroutine that prints out relevant data for the phase, _ _ _PRT; and a subroutine that allows input and computed data to be plotted, _ _ _PLT.

In addition to the mainline subroutines, there is a group of subroutines known as "kernel" subroutines. These subroutines perform generation tasks specific to that phase. Figure 8 shows a conceptual flowchart for a typical GENERATOR phase. Kernel subroutines may be called at any time in the phase, whereas mainline subroutines must be called in the proper order.

All GENERATOR phases may call Utility subroutines from any subroutine. (Utility subroutines are defined as those subroutines which are common to more than one phase in a phase group). However, certain utilities are called from specific locations or only at specific times. For example, ENTER/EXIT utilities are the first and last executable statements in each mainline subroutine; only _ _ _INP and _ _ _CHK call ERROR utilities; INPUT utilities are concentrated in _ _ _INP, etc. A list of typical utilities are shown in Table 4.

Each PROCESSOR phase group is composed of eight phases -- one phase less than the GENERATOR phase group (there is one phase corresponding to each of the GENERATOR phases except the CMN phase). For STEALTH, all phases

are mandatory and all phases must be executed in a precise order. Figure 9 displays the PROCESSOR phase group flowchart for serial STEALTH.

In each PROCESSOR phase there is at least one mainline subroutine called _ _ _PRO. It is analogous to _ _ _GEN in the GENERATOR, where _ _ _ is the phase name identifier. In the GPT and ZON phases there are two other mainline subroutines, _ _ _OLD and _ _ _NEW. Subroutine _ _ _OLD transfers data at times n-1/2 and n from array variable storage locations to storage locations in nonarray variables. These nonarray variables are used in physics calculations to update "old" values of variables (times n-1/2 and n) to "new" values of variables (times n+1/2 and n+1). Subroutine _ _ _NEW then transfers the data at times n+1/2 and n+1 from nonarray storage locations to appropriate array storage locations.

All other subroutines in the PROCESSOR are kernel or utility subroutines. A special group of kernel subroutines, which describes material model response characteristics, is found in the ZON phase of the PROCESSOR.

For a vector mode version of STEALTH, the PROCESSOR would take a slightly different form at the subroutine level and would require a different phase calling order. However, the overall design concepts would remain.

CONCLUSION

Implementing a structured architecture for the STEALTH code system has made it (1) easier to debug and modify logic, (2) simpler for new users to learn how the codes work, and (3) ideal for maintaining versions on different computer hardware. The "theoretical elements of analysis", which are the basis for program design in STEALTH, have proven to be a useful concept for solving general nonlinear equations approximated by the Wilkins explicit finite-difference solution technique.

REFERENCES

1. Ronald Hofmann; "STEALTH, A Lagrange Explicit Finite-Difference Code of Solids, Structural, and Thermo-hydraulic Analysis", EPRI NP-260, Vols. 1-3, Electric Power Research Institute, Palo Alto, California, August 1976. Prepared by Science Applications, Inc., San Leandro, California.

2. Mark L. Wilkins; "Calculation of Elastic-Plastic Flow", UCRL-7322, Lawrence Livermore Laboratory, Livermore, California, April 19, 1963. Also in Methods in Computational Physics, Berni Alder, Sidney Fernbach, and Manual Rotenberg, eds., Vol. 3, Fundamental Methods in Hydrodynamics, Academic Press, New York, 1964 (211-263).

3. Walter Herrmann; "A Lagrangian Finite-Difference Method for Two-Dimensional Motion Including Material Strength", WL-TR-64-107 (AD 609 523), Air Force Weapons Laboratory, New Mexico, November 1964.

TABLE 1. THEORETICAL ELEMENTS OF A PHYSICAL SYSTEM

Conservation laws	physical principles governing all motion
Boundary conditions	geometric constraints and boundary values
Initial conditions	initial state of things
Constitutive relations	material models

TABLE 2. ELEMENTS OF THEORY, CODE STRUCTURE, AND DOCUMENTATION AFFECTING USER ORIENTATION

THEORY

1. Physical Laws
2. Mathematical equations
3. Numerical equations

CODE STRUCTURE	DOCUMENTATION
1. Programming practices	1. Input manual
2. Modular structure	2. Flow charts
3. Input/Output	3. Vocabulary

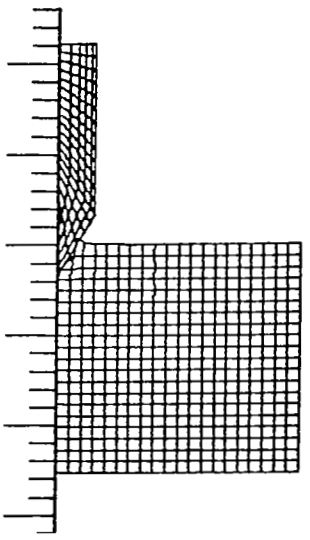
TABLE 3. CORRESPONDENCE BETWEEN OPTIONAL GENERATOR PHASES AND CONTINUUM MECHANICS ELEMENTS OF ANALYSIS

Constitutive Equations (material models)	MATGEN
Boundary Conditions (control volume definition)	
geometric constraints	GPTGEN
boundary values	BDYGEN
Initial Conditions	ZONGEN
Conservation Equations	TIMGEN
Output	EDTGEN

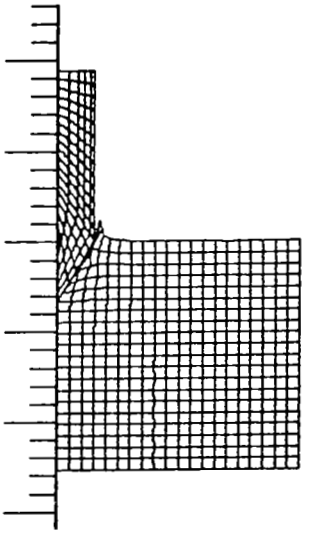
TABLE 4. TYPICAL UTILITY PHASE SUBROUTINES

SYSTEMS	OUTPUT	ENTER/EXIT	ARITHMETIC
RUNDAT	PGEHDG	SBRENT	FNCONE
RUNTIM	TIMHDG	PHSENT	MYFNO
	PHSHDG	SBREXT	FNCTWO
INPUT	INPHDG*	PHSEXT	MYFNT
	CHKHDG	GENEXT	FDVONE
CRDTTL*	PRTHDG	PROEXT	MYFDO
CRDINP*	PLTHDG	ERREXT	
CRDPRT*			
LIBTTL			
LIBINP			
LIBPRT		ERROR	MISCELLANEOUS
KBDTTL*			
KBDINP*		CHRERR	INPDGT
KBDPRT		FLDERR	FLDCHK*
REWFLS		LIMERR	CNVDTA
RINREC		MDLERR	PHSCHK
GETDTA		RGEERR	
WOTREC		SBRERR	
PUTDTA		TYPERR	

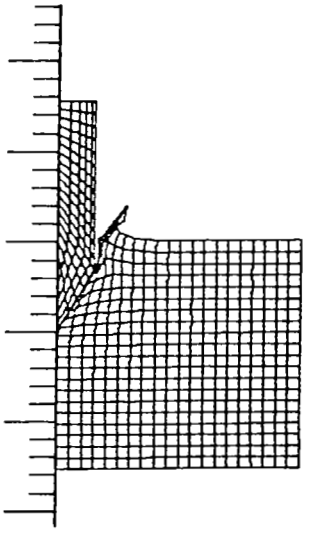
* Uses standard input record format



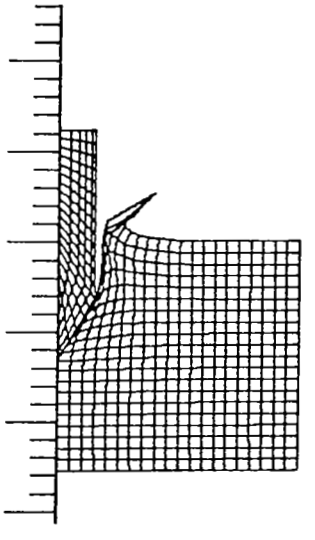
t = 4



t = 8



t = 12



t = 16

Figure 1.- Penetration of steel projectile into aluminum target using STEALTH 2D.

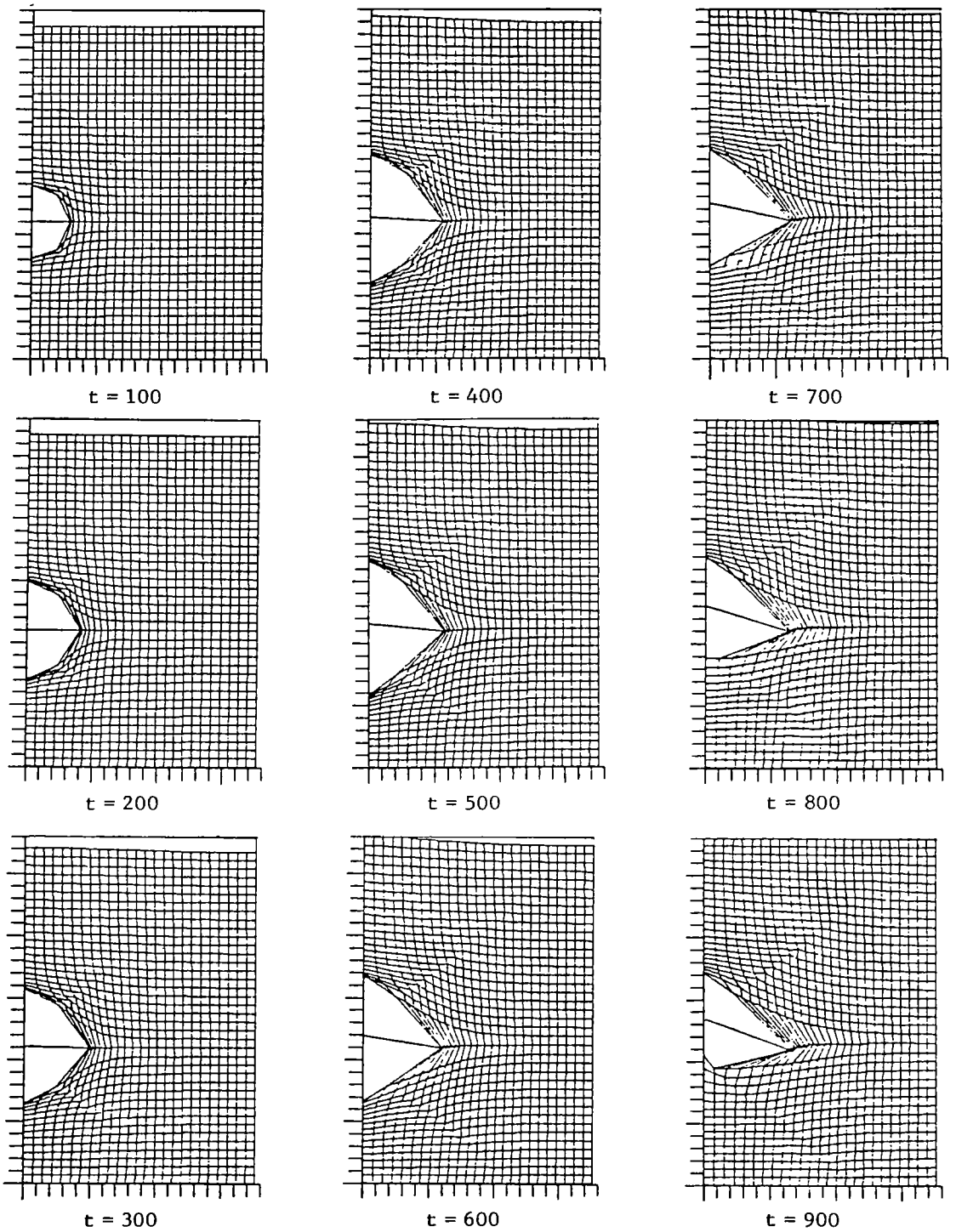


Figure 2.- Axisymmetric STEALTH 2D simulations of energy release in fluid contained in overstrong cylindrical container.

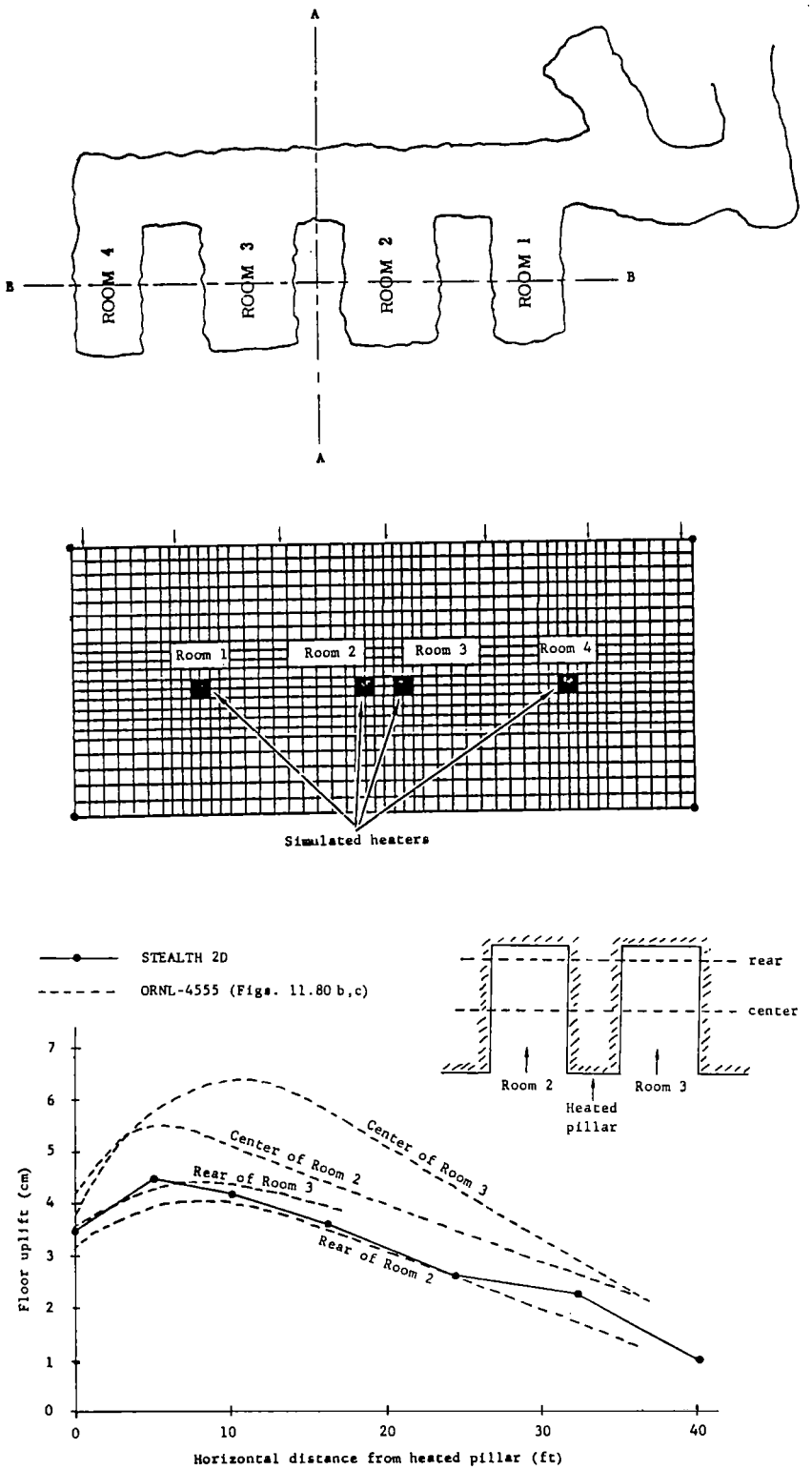


Figure 3.- Nuclear waste repository simulation using STEALTH 2D.

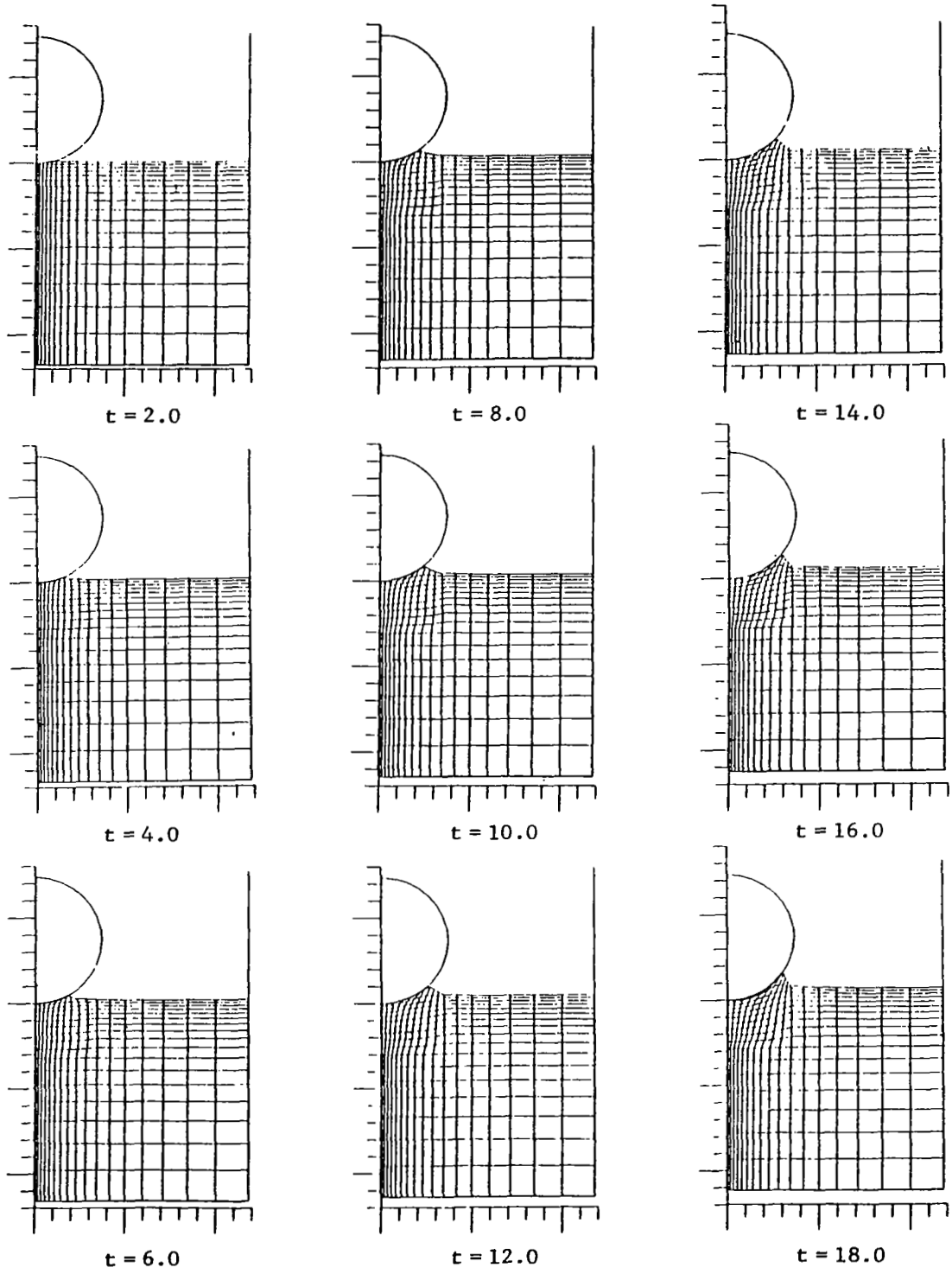


Figure 4.- Translational symmetry STEALTH 2D simulation of long rigid pipe being pushed down into tank of water.

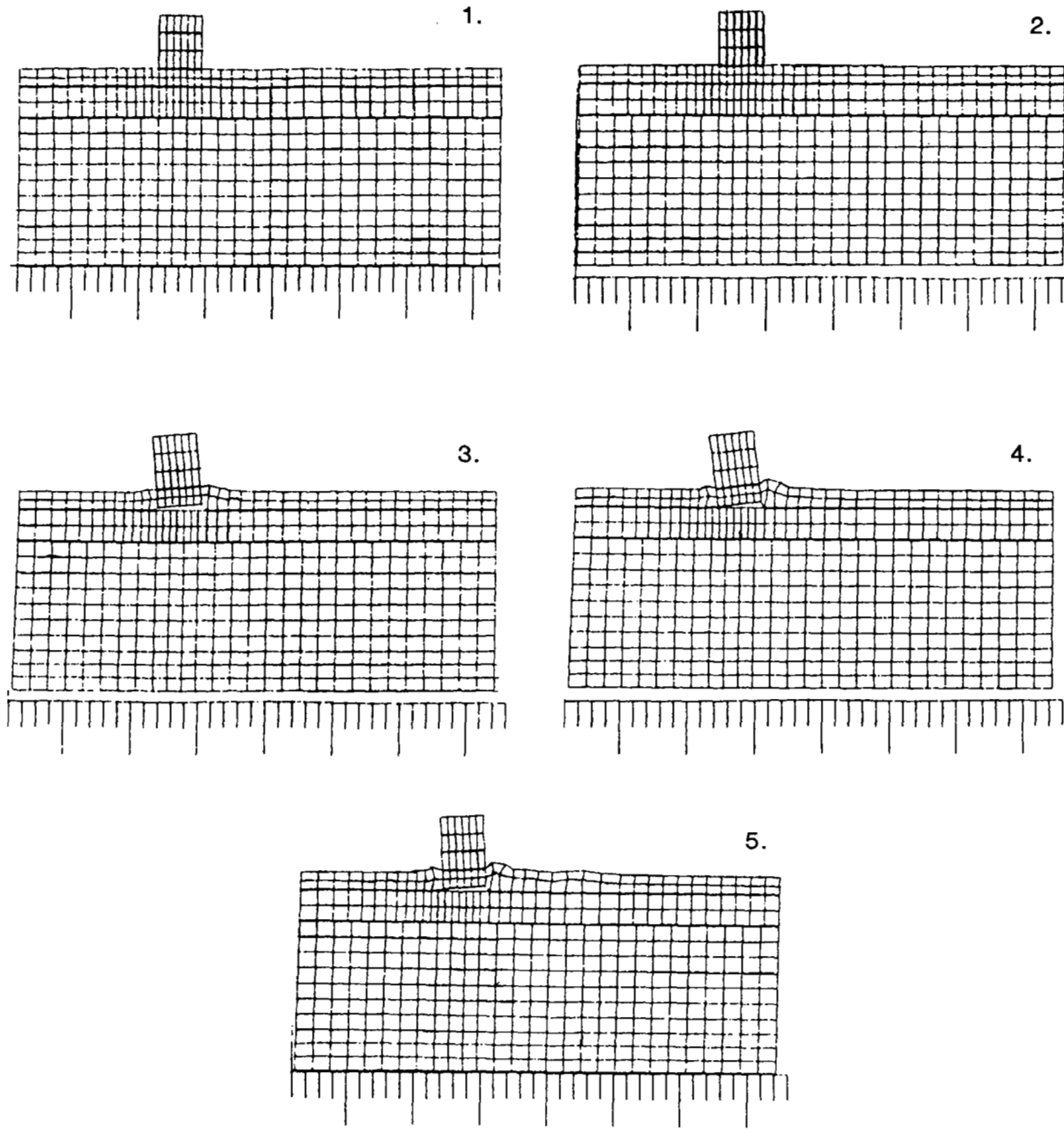


Figure 5.- Rocking of building on soil island using STEALTH 2D.

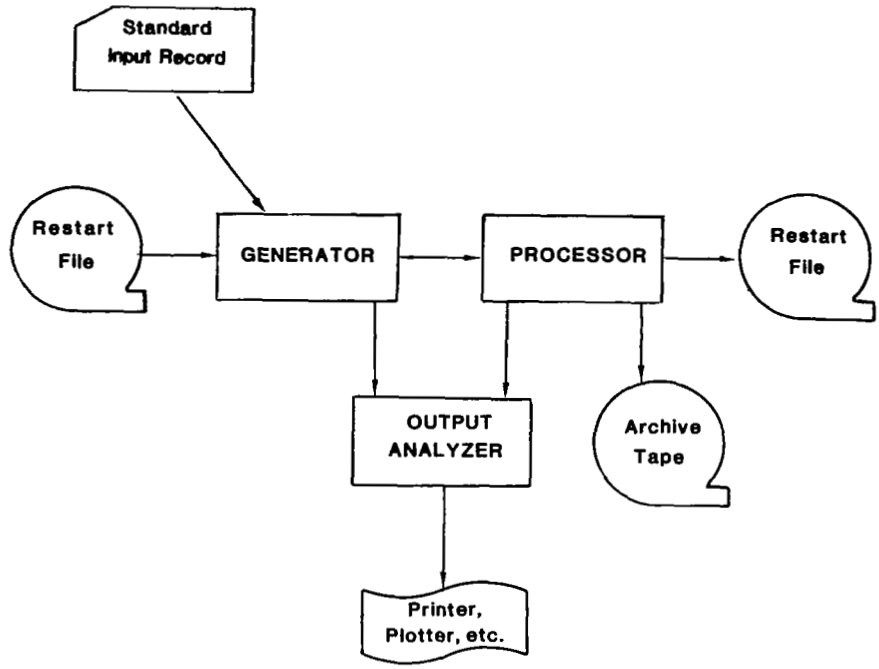


Figure 6.- Phase group flowchart.

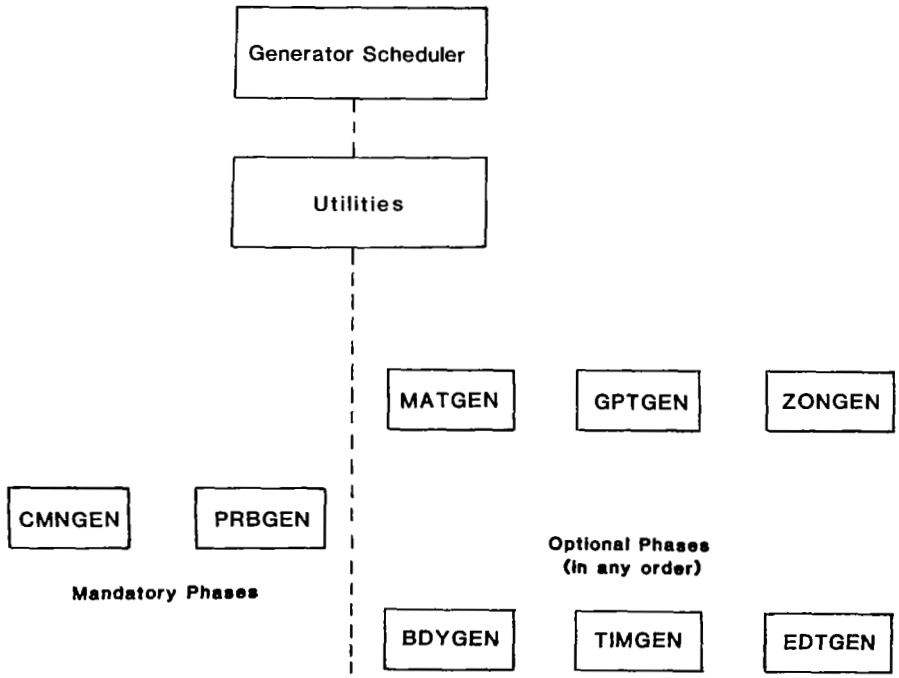


Figure 7.- GENERATOR phase group flowchart.

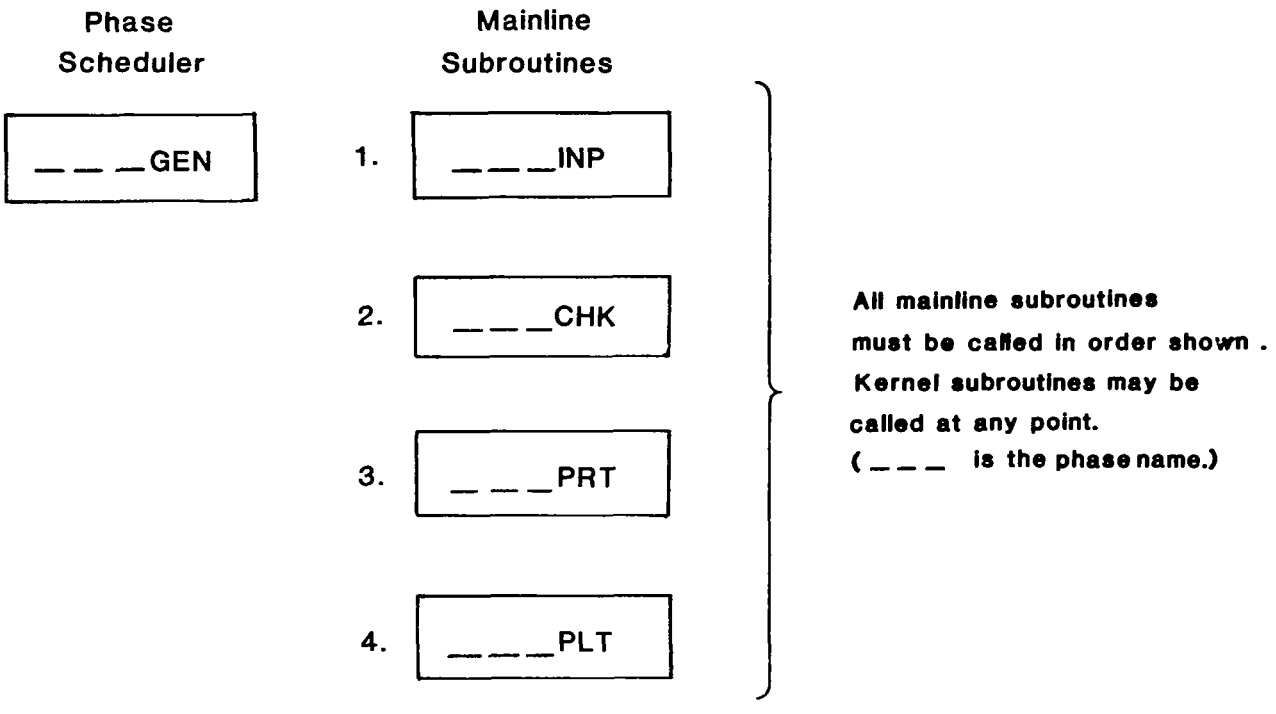


Figure 8.- Typical GENERATOR phase structure.

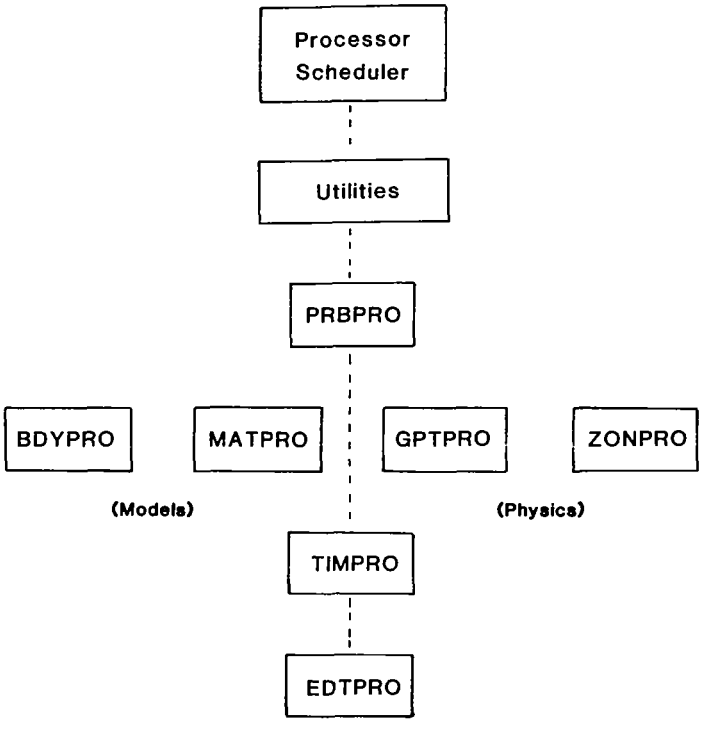


Figure 9.- PROCESSOR phase group flowchart.