CONF-860722--14

CONSOLIDATED FUEL REPROCESSING PROGRAM

FORPS: A FORTH-Based Production System and Its Application to a Real-Time Robot Control Problem

CONF-860722--14

DE86 005426

Christopher J. Matheus
University of Illinois
Urbana, Illinois

H. Lee Martin
Instrumentation and Controls Division
Oak Ridge National Laboratory*
Oak Ridge, Tennessee

Paper for presentation
at the
Computers in Engineering Conference
Chicago, Illinois
July 20, 1986

MASTER

*Operated by Martin Marietta Energy Systems, Inc., for the U.S. Department of Energy.

[illegible]

CONSOLIDATED FUEL REPROCESSING PROGRAM

FORPS: A FORTH-Based Production System and Its Application to a Real-Time Robot Control Problem[*]

Christopher J. Matheus
University of Illinois
Urbana, Illinois

H. Lee Martin
Instrumentation and Controls Division
Oak Ridge National Laboratory
Oak Ridge, Tennessee

Abstract

A simple yet very powerful system has been developed that merges the artificial intelligence qualities of a production system with the real-time control capabilities of FORTH. FORPS (FORTH-based Production System) offers the advantages of intelligent, rule-based control in a small package offering high speed, extensibility, and simplicity. A practical example of the system is presented in the development of an obstacle avoidance program to aid in controlling an overhead manipulator transport system. Several other potential applications to the area of control are discussed.

[*]Research sponsored by the Office of Facilities, Fuel Cycle, and Test Programs, U.S. Department of Energy, under Contract No. DE-AC05-84OR21400.

Introduction

This paper presents a method for combining the real-time power and versatility of FORTH with the intelligent decision-making structure of a production rule system. This hybrid system is called FORPS (FORTH-based Production System). It is concise, yet it provides the potential deductive power of a rule production system. Since FORPS is written in FORTH, it maintains the many advantages of a FORTH programming environment. An overview of the FORPS design and a source code listing are provided in Appendix A to give the interested reader the opportunity to use FORPS. The first section of this paper presents a brief description of production systems, followed by a discussion of some of the features of FORTH for real-time applications. An example is then presented in which FORPS is applied to the problem of obstacle avoidance in the transportation of a mobile manipulation system. This particular manipulator system is the Advanced Servomanipulator (ASM) developed at the Oak Ridge National Laboratory (ORNL) for the Consolidated Fuel Reprocessing Program (CFRP) [1]. The potential for applying FORPS to other real-time problems is also explored. While the program examples and FORPS are written in FORTH, previous exposure to FORTH is not essential to understanding the majority of this paper.

What is a Production System?

Of the several artificial intelligence (AI) programming techniques available, production systems have perhaps gained the widest acceptance. A production system is fundamentally a collection of condition-action rules. An individual production consists of a conditional clause and an action statement, usually bound together in an IF..THEN structure. Following is an example of a production as it might appear written in plain English [2]:

```
Production 1
If   it has feathers
     and it flies
     and it lays eggs
Then it is a bird
```

This rule states that if "it" (the object in question) has feathers, flies, and lays eggs, then make the deduction that it is a bird. A production system implemented for a specific area of knowledge is often referred to as an expert system. A system of this nature might consist of several hundred similar rules.

A major advantage of production systems is that they are easily extended to accommodate new information. Production systems consider all rules equally by means of the inference engine, a program that scans through the set of all rules and executes the action portion of a rule whenever its conditional clause is satisfied. If, however, more than one rule is satisfied at the same time, a method of "conflict resolution" (based on a set of priorities) determines which rule to "fire" (i.e., execute). This method of evaluation gives relative independence to the rules, making it easy to add new knowledge to a production system even after installation. Many production systems are in fact designed to be developed incrementally through the gradual addition of new rules [2,3].

A major drawback of traditional production systems is the amount of time required to arrive at a final conclusion. Other major shortcomings are the limited input/output capability of production systems and the need for specialized machines to run them. An unfortunate consequence of these limitations is that production systems are often unable to solve real-time, real-world problems.

What is FORTH?

Charles H. Moore began developing the FORTH programming language during the late sixties. At that time, "traditional languages did not provide the power, ease, or flexibility" [4] he needed, and he began development of a new programming tool that eventually evolved into FORTH. From the start, the concept of FORTH was dominated by Moore's principle, "keep it simple."

A significant advantage of FORTH is that it provides full power and control over all of the machine's capabilities. It is possible to access any memory location simply by specifying its address. This applies to all input/output ports, making it very easy and efficient to read/write to peripheral devices such as A/D converters, timers, counters, and switches. This ability is crucial for the sensing necessary to real-time control. Control over the machine and its peripherals, combined with its speed and programming ease, makes FORTH a logical choice for real-time control applications.

Example of a Real-time FORTH Application

A prime example of a real-time application of FORTH is the Advanced Integrated Maintenance System (AIMS) being developed at ORNL (see Fig. 1). A pair of digitally controlled master/slave servomanipulators--the Advanced Servomanipulators (ASM)--are being built for use in a teleoperated remote handling system (see Fig. 2). The AIMS system is a prototype for a system ultimately intended for application in a nuclear fuel reprocessing facility, with the slave manipulators mounted on an overhead transport system within a radioactive "hot cell" and the master operated from a remotely located control facility.

Four software packages are being developed for AIMS [5]: a servo control package, a transporter and television camera package, a communication package, and a man-machine interface (MMI). The servo package controls the servo loops operating between the master and slave joint motors. This software, which must run at a frequency of at least 100 Hz, is handled by eight Motorola 68000 microprocessors. The camera and transporter package controls positioning of the overhead transporter and the multiple in-cell cameras. The transporter allows the ASM to be moved to any desired location within the cell, and the cameras provide visual monitoring of maintenance operations. The communication package provides information transfer between devices via a local area network. The MMI is the link between the operator and the other three control packages. It runs on two Motorola 68000 processors to perform command selection and provide system information and status for the operator. All four software packages must run in real time, and all are written in FORTH. A more detailed description of the AIMS control system can be found in reference 5.

The Need for Artificial Intelligence

In many areas of AIMS software, the addition of some form of AI would substantially improve overall system operation. For example, the MMI is used by the operator to perform functions such as changing task modes, analyzing system status and statistics, diagnosing faults and failures, and controlling the camera and overhead transport packages. Currently, the operator is required to mentally process a significant amount of this information. It would be advantageous to have the MMI ease the operator's workload by performing a larger percentage of intelligent processing of system information for tasks such as obstacle avoidance, automatic camera tracking, and system diagnostics. With greater machine intelligence, operation of the system should become easier, more efficient, and safer. It is important, however, that this added intelligence operate in real time without slowing system response; otherwise, such efforts will hinder rather than help the operator.

Why FORPS?

To implement traditional AI programming techniques on the AIMS through a standard production system would be difficult, time consuming, and costly due to the time and effort invested in AIMS' FORTH-based software. It would be more desirable to develop a system that would allow use of AI production system techniques without rewriting major portions of existing code. In other words, write production systems from within FORTH. Because FORTH is extensible and is easily customized to any application, the logical solution was to extend FORTH to recognize production system rule structures. FORPS is based on this concept.

The FORPS Design

The design philosophy of FORPS is based on the criteria that it should be small, simple, and fast enough to be applicable to real-time problems. Further, it should provide the power of a basic production system while maintaining FORTH's approach to programming (e.g., extensible, flexible, fast, etc.). One important goal was that it be possible to execute any legal FORTH word at any point within a production rule. (In a real-time control situation it is desirable to be able to perform an operation--such as scanning an I/O port--within the conditional clause of a rule.) The ability to execute any FORTH word from within either the conditional phase or the action phase of a rule gives the programmer a higher degree of flexibility and power, two FORTH qualities that FORPS attempts to preserve.

FORPS is intentionally very simple and short in order to keep it fast for use in real-time applications. It consists of only five rule-defining words and the inference engine, but the fundamental components and potential power of a full-fledged production system are completely intact. The object code is less than 3 Kbytes. It is possible to add enhancements to a minimum FORPS system if a special need becomes apparent. As an example of its theoretical usefulness, FORPS has been used to solve the classic AI problem, "The Towers of Hanoi."

Rule Definition Words

The main task of writing a production system is defining the rules. FORPS contains only five rule-defining words: **RULE:**, **PRIORITY:**, ***IF***, ***THEN***, and ***END***. Following is an example of a rule definition:

```
RULE:  IS-A-BIRD       PRIORITY:  1
*IF* HAS-FEATHERS
     FLIES
     LAYS-EGGS
*THEN*. "The animal is a bird."
*END*
```

Consider the construction of this rule. RULE: takes the next word, IS-A-BIRD, as the name of the rule being defined. PRIORITY: sets up a relative priority for rules in case the conditional clause of more than one rule is satisfied at the same time. In such an event, the rule with the higher priority is executed. The use of PRIORITY: is strictly optional. When PRIORITY: is not used, a rule receives the lowest default priority, zero.

IF begins the conditional portion of the rule. Each word that follows *IF* becomes a part of the conditional clause that will be executed when the inference engine scans conditionals. These words must be predefined, executable FORTH words. In the above rule, HAS-FEATHERS, FLIES, and LAYS-EGGS would all be FORTH-defined words that presumably would return values according to the state of the system (i.e., the data in memory and the conditions of the I/O ports). For example, HAS-FEATHERS would most likely return a true value if the animal in question has feathers. This value might come either directly from a variable in memory or from a peripheral device that performs a test to see if the animal has this quality.

The word *THEN* marks the beginning of the action portion of the rule. The words that follow must also be predefined, executable FORTH words, because they will be executed by the inference engine whenever the conditional clause is satisfied and the rule is of highest priority. The word *END* simply marks the end of the rule definition.

Additional rules would be defined in a manner similar to the example above. All rules in the system would then be loaded by the standard FORTH compiler. To run the system, the inference engine is invoked with the "FORPS" command and sequentially scans the rules to determine which conditional clauses are satisfied. The satisfied rule with the highest priority is then executed, and the inference engine repeats this cyclical process until none of the conditional clauses are satisfiable or until an explicit HALT command is executed.

Application to Obstacle Avoidance

A practical application of FORPS is the real-time problem of obstacle avoidance. The ASM is transported through its cell environment by way of an overhead transport system, which an operator controls with a joystick while observing the system's motion through several television cameras. An additional graphic cell map is computer generated as shown in Fig. 3. The cell map displays the manipulator's position within the cell as well as the location of potential obstacles, which are color coded according to their height relative to the height of the manipulator: Red obstacles are higher than the manipulator and represent absolute obstructions; Yellow obstacles are lower than but very nearly the height of the manipulator and are considered potential hazards. Information concerning the height and location of all objects is statically maintained within the cell map's data base. Although there are obvious problems with a static representation of objects within the cell, their solution is beyond the present scope of this work.

A production-based program consisting of seven rules has been written to assist the operator in obstacle avoidance during manipulator transportation (see Appendix B). This obstacle avoidance system overlays the existing transporter control software. Its function is to analyze the operator's direction requests and prevent the execution of requests that would result in collision with an obstacle. The production system scans data stored in the cell-map data base to determine if the manipulator is being directed toward an object of potentially dangerous height. If the manipulator would come close to hitting an obstacle, a warning is displayed, but the operator is allowed to proceed in the desired direction. If the path is blocked, the transporter is prevented from proceeding in the requested direction, and a "PATH OBSTRUCTED" message is displayed for the operator. In certain instances it may be necessary to proceed cautiously near an obstacle; an "override" switch is available to the operator to override the system's control. Walls are shown in white, and the operator is never allowed to command the transporter to collide with a wall.

Some important observations should be made concerning this application. First, the obstacle avoidance system was written using existing FORTH code and data structures. Words to move the transporter and to access and display the cell map's data had already been written for an unintelligent cell-map/transporter system. Because these words were already in use, design and operation of the obstacle avoidance production system was accomplished in a very short time (roughly four hours). Second, the system operates in real time. There is no response difference between this and the previous, unintelligent cell-map/transporter system. Third, this simple system may be extended easily by adding rules to increase its intelligence. One possible enhancement is the addition of an automated path-finding routine in which the operator specifies a destination and the program autonomously transports the manipulator safely through the cell to that location.

A major advantage of using FORPS to solve this problem is the ease with which high-level condition-action rules can be implemented. The obstacle avoidance problem lends itself to a definition in terms of production rules because one naturally orders the significant events in terms of rules such as "if there is no obstacle, then move the transporter as requested," and "if there is a wall, then prevent movement and print a warning message." These if/then constructs obviously could be implemented with conventional programming techniques, but FORPS provides a flexible, orderly framework which allows efficient program development. It is also likely that a conventional approach would execute more slowly and be prone to logical errors.

Other Applications

The obstacle avoidance system is just one of many potential applications for which FORPS is ideally suited. Another practical application of FORPS being considered at ORNL is ASM system diagnostics, in which an expert system would perform continuous diagnostic tests on the ASM. The many potential causes of faults and failures include mechanical breakdowns in the gear driven arms, electrical faults in the amplifiers and wiring, and logical failures in the processors controlling the ASM. An expert diagnostics program is envisioned that would run on a dedicated processor, continually monitoring the system's status. Upon detection of an error, the arms would be shut down and specific diagnostic routines would be performed. Since the monitoring of system errors amounts to little more than the continual cycling through of several tests, a FORPS production system would be an ideal way to construct a diagnostics system.

Similar possibilities exist for applying FORPS methodology to problems such as television camera control, overhead crane operation, coordinated arm tasks, and manipulator maintenance. FORPS also could be extended to development of expert systems on personal computers.

Summary

FORPS is a production system developed in a FORTH programming environment. It is very simple and fast, yet maintains the many advantages of FORTH. Its simplicity, makes possible real-time execution speeds. FORPS potentially possesses the power of a traditional production system and, since it is extensible, additional capabilities can be added if such needs become evident. Furthermore, the experienced FORTH programmer can learn and implement it easily.

Because of its unique qualities, FORPS makes possible the application of rule-based AI programming techniques to real-time control problems such as the ASM obstacle avoidance system, which runs in real time to assist the operator in transporting the ASM through its obstructed environment. Similar opportunities exist for applying FORPS to other FORTH-based systems in which it may be necessary or desirable to add a degree of intelligent control. By combining the real-time input/output capabilities of FORTH with a rule production structure, a tool is now available to apply AI techniques to real-time, real-world problems.

References

[1] Kuban, D. P., and H. L. Martin, "An Advanced Remotely Maintainable Force-Reflecting Servomanipulator Concept," **Proceedings, 1984 ANS Topical Meeting on Robotics and Remote Handling in Hostile Environments**, 407-15 (1984).

[2] Winston, P. H., **Artificial Intelligence** (2d Ed.), Addison-Wesley, 1984.

[3] Barr, A., and E. A. Feigenbaum (Eds.), **The Handbook of Artificial Intelligence**, Vol. 1, William Kaufman, Inc., 190-199 (1981-82).

[4] Brodie, Leo, **Starting FORTH**, Prentice-Hall, 1981.

[5] Martin, H. L., et al., "Control and Electronic Subsystems for the Advanced Servomanipulator," **Proceedings, 1984 ANS Topical Meeting on Robotics and Remote Handling in Hostile Environments**, 417-24 (1984).

APPENDIX A

FORPS Source Code

```
460 LIST

   ( FORPS -- a FORth-based Production System )       ( CJM 8/19/85)
  1 3 +THRU
  EXIT

  FORPS is under the copywrite of Martin Marietta Energy Systems.
       It is in the public domain and may be used freely
       as long as no false claims are made to its authorship.
       Author:   Christopher J. Matheus
                 University of Illinois
                 222 Digital Computer Lab
                 1304 W. Springfield Ave.
                 Urbana, IL   61801

  This software was developed at Oak Ridge National Laboratory,
  Oak Ridge, TN, under the Consolidated Fuel Reprocessing Program.

461 LIST

     ( FORPS constants and variables )               ( CJM*8/15/85)
  10 CONSTANT MAX-#RULES    16 CONSTANT RULE-LEN
  VARIABLE NO-ACTIVITY    VARIABLE 'SP-IF         VARIABLE 'NOOP
  VARIABLE >RULE-TABLE    VARIABLE >LAST-RULE     VARIABLE CYCLE
  VARIABLE HIGH-PRI       VARIABLE BEST-ACTIVE-RULE
  CREATE RULE-TABLE       MAX-#RULES RULE-LEN * ALLOT

  : >ACTION    ( a -a)  4 + ;
  : >FIRE-CELL    ( a -a)  8 + ;
  : >PRIORITY    ( a -a)  12 + ;
  : HALT   NO-ACTIVITY TRUE ;
  : *ERROR*    1 ABORT" NO RULES LOADED" ;
  : *RESET-FORPS*   RULE-TABLE DUP >RULE-TABLE !  MAX-#RULES
     RULE-LEN * ERASE  ['] *ERROR* RULE-TABLE ! ;  *RESET-FORPS*
  : NOOP ;    ' NOOP 'NOOP !    ' NOOP 4- @ CONSTANT COLON-CFA


462 LIST

     ( FORPS rule defining words )                   ( CJM*8/15/85)
  : COND-PFA!    ( a)  HERE >RULE-TABLE @ ! ;
  : ACTION-PFA!    ( a)  HERE >RULE-TABLE @ >ACTION ! ;
  : RULE:    >RULE-TABLE RULE-LEN / MAX-#RULES = ABORT" no room"
     CURRENT W@ CONTEXT W!  CREATE  COND-PFA!  -4 ALLOT
     COLON-CFA ,    SMUDGE ] ;
  : PRIORITY:    >RULE-TABLE @  -' IF NUMBER ELSE DROP EXECUTE THEN
     SWAP >PRIORITY W! ;  IMMEDIATE
  : *if*    -1 'S 4- 'SP-IF ! ;
  : *IF*    >RULE-TABLE @ >FIRE-CELL [COMPILE] LITERAL
     COMPILE *if* ; IMMEDIATE
  : *then*    ( *n)  -1  'SP-IF @ 'S DO AND 4 +LOOP  SWAP ! ;
  : *THEN*    COMPILE *then* COMPILE EXIT  COLON-CFA ,
     ACTION-PFA!  ; IMMEDIATE
  : *END*    RULE-LEN >RULE-TABLE +!  COMPILE EXIT  SMUDGE
     R> DROP ;    IMMEDIATE

463 LIST

     ( FORPS Inference engine )                      ( CJM*8/15/85)
  : SET-DEFAULT    -1 HIGH-PRI !  'NOOP BEST-ACTIVE-RULE ! ;
  : RT-LIMITS    ( -n n)  >LAST-RULE @  RULE-TABLE ;
  : CLEAR-FIRES   RT-LIMITS DO 0 I >FIRE-CELL !  RULE-LEN +LOOP ;
  : TEST-RULE-CONDS   RT-LIMITS DO I @EXECUTE  RULE-LEN +LOOP ;
  : SELECT-BEST-RULE   NO-ACTIVITY TRUE  SET-DEFAULT
    RT-LIMITS DO I DUP >FIRE-CELL @
                IF DUP >PRIORITY W@ DUP HIGH-PRI @ >
                   IF HIGH-PRI !  >ACTION BEST-ACTIVE-RULE !
                      NO-ACTIVITY FALSE
                   ELSE 2DROP THEN
                ELSE DROP THEN RULE-LEN +LOOP ;
  : FIRE-RULE    BEST-ACTIVE-RULE @  @EXECUTE ;
  : FORPS    >RULE-TABLE @ 4- >LAST-RULE !  0 CYCLE !
    BEGIN 1 CYCLE +!  CLEAR-FIRES  TEST-RULE-CONDS
            SELECT-BEST-RULE  FIRE-RULE  NO-ACTIVITY @ UNTIL ;
```

1060 LIST

```
              FORPS -- FORth-based Production System
  FORPS is a simple Production System designed to take full
advantage of the powers of FORTH.  Basic production-like rules
are constructed using the words RULE:, *IF*, *THEN* and *END*.
These rules are put into a table of the following format:
        COND. pfa  !  ACTION pfa  !  FIRE cell  !  PRIORITY value
        four bytes !  four bytes  !  four bytes !  two bytes
Rule 1      *            *             *            *
   :        :            :             :            :
Rule n      *            *             *            *
   The COND. pfa is the pfa of the conditional portion of the
rule and the ACTION pfa is the action portion's pfa.  The FIRE
cell holds the result of the conditional pfa's execution.
PRIORITY is a number between 0 and 65534 used in the choosing
of the best-active-rule during "conflict-resolution".

```

1061 LIST

```
FORPS constants and variables
MAX-#RULES maximun nuber of rules allowed in table
RULE-LEN length of a single rule table entry
NO-ACTIVITY true if no rules fired during the cycle
'SP-IF saves the parm. stack addr. at start of cond.
$RULE-TABLE pointer into the rule table
>LAST-RULE address of last rule in table
CYCLE number of cycles executed
HIGH-PRI highest priority of all rules fired
BEST-ACTIVE-RULE action pfa of highest priority active rule
>ACTION, >FIRE_FLAG, >PRIORITY offsets into rule-table
HALT sets NO_ACTIVITY to true -- causes FORPS to terminate
*ERROR* stored as 1st rule when RESET - aborts from FORPS
*RESET* clears rule-table and loads *ERROR* as 1st rule
NOOP no-operation, used as default BEST-ACTIVE-RULE
COLON-CFA cfa of : -- the contents of the cfa of : words
```

1062 LIST

```
FORPS rule words
COND-PFA!  stores condition-pfa of rule into table
ACTION-PFA!  stores action-pfa of rule into table
RULE:  adds a rule to the dictionary -- creates two words
  with one head: first is cond word, second is action word
PRIORITY:  loads the rule's priority cell with the value of
  the next word in the input stream (usually a number)
*if*  runtime version of *IF*, saves stack pointer and puts
  a -1 on stack for subsiquent use by *then*
*IF*  compiles as a literal a pointer to the rule's
  fire-cell (for use by *then*), and compiles *if*
*then*  runtime version of *THEN* -- AND's the cond. stack
  items and stores the result in the rule's FIRE cell
*THEN*  compiles *then*, compiles EXIT to stop cond. word,
  and compiles COLON-CFA to begin rule's action word
*END*  incrms. rule counter and ends rule compilation
```

1063 LIST

```
Inference engine
SET-DEFAULT sets BEST-ACTIVE-RULE to NOOP, clears priority
RT-LIMITS  puts the rule-table's addr. limits on the stack
CLEAR-FIRES clears the fire flags of all rules
TEST-RULE-CONDS executes in order each rules condition pfa
EXECUTE-ACTIVE-RULES executes the action of the highest
  priority rule which has fired.  If no rules fired NO-ACTIVITY
  will be set to true and the FORPS loop will terminate
FIRE-RULE  fires the BEST-ACTIVE-RULE
FORPS  the main inference loop of the production system.
  After resetting #RULES and CYCLES, the inference loop is
  entered and continues until a cycle passes in which no
  rules have fired.  The inference loop has three functions: it
  increments the cycle counter, clears all fire cells,  tests
  the conditional clauses,  and executes the highest priority,
  active rule.
```

APPENDIX B

Obstacle Avoidance Productions

```
380 LIST

( Cellmap  PS constants & variables )                ( CJM*8/09/85)
 1 CONSTANT WALL    7 CONSTANT LOW-OBS    2 CONSTANT HIGH-OBS
1CLOSE 40 + CONSTANT ROBOT-POS

VARIABLE STARTED    VARIABLE NEWDIR    WVARIABLE DIR-CONTENTS
VARIABLE XDIR       VARIABLE YDIR      VARIABLE ZDIR
VARIABLE CELL-EXIT VARIABLE CPAD-STATE
VARIABLE OVERIDE


: TRAN_CELL_MAP    STARTED FALSE FORPS ;
                   ( TRAN_CELL_MAP simply executes
                     the production system rule set )




381 LIST

    ( Cellmap input words )  HEX                        ( CJM*8/09/85)

: ?MENU-SEL PAD_VALUE @ 80000000 AND NOT ;
: ?CELL-PAD PAD_VALUE @ 3FFFFFFF AND 5 = ;
: ?CELL-CMD PAD_VALUE @ DUP 3FFFFFFF AND PAD_VALUE !
    40000000 AND ;
:-CPAD-ALLOW 80000000 PAD_VALUE +! ;
: READ-JOYSTICK ( --n/n)  FFFF40 C@ 1F AND 1F XOR
    FFFF42 C@ 3 AND  3 XOR ;
: READ-OVERIDE    ( -n)  FFFF42 C@  F AND DUP 7 = SWAP 8. = OR
   DUP IF OVERIDE TRUE THEN PAUSE ;






382 LIST

    ( Cellmap menu words )                             ( CJM*8/09/85)
: CPAD-ACCENT -200 100 MOVABS 1 PRMFIL WHITE COMP
  135 35 RECREL UNCOMP ;
: ACCENT-CPAD ?CELL-PAD IF CPAD-STATE @ NOT
  IF CPAD-ACCENT 1 CPAD-STATE ! THEN
  ELSE CPAD-STATE @ IF CPAD-ACCENT 0 CPAD-STATE ! THEN THEN ;
: CPAD-CHECK    ?MENU-SEL IF ACCENT-CPAD ?CELL-CMD IF ?CELL-PAD
   IF 1 CELL-EXIT ! THEN THEN CPAD-ALLOW THEN ;
: CELL-PAD -200 100 MOVABS CYAN 32 10 MOVREL
  3 0 TEXTC V." EXIT" 0 0 TEXTC ;
: CELL-MAP-INIT 1 PRMFIL 0 CELL-EXIT ! 0 CPAD-STATE !
   BLACK FLOOD CELL-ORG CELL-CHAR RESTARER CELL-PAD 2 0 TEXTC
   CUR_STOP CPAD-ALLOW ;
: CELL-MAP-EXIT  BLACK FLOOD  3 0 0 CLOAD 4 0 0 CLOAD CR
   CUR_REPORT STAT_INIT HEAD_INIT  0 1 MENU  STATUS_HANDLER ;


383 LIST

    ( Obstacle detection words )                       ( CJM*8/09/85)
: WORST    ( n n - n)  OVER 1 = OVER 1 = OR IF 2DROP 1 ELSE
   OVER 2 = OVER 2 = OR IF 2DROP 2 ELSE
          7 = SWAP 7 = OR IF        7 ELSE 0  THEN THEN THEN ;
: SCAN-DIR    ( - n) XDIR @  YDIR @
   2DUP  17 = SWAP  11 *   + ROBOT-POS + C@ >R
   2DUP  18 = SWAP   2 *   + ROBOT-POS + C@ >R
         19 = SWAP  -7 *   + ROBOT-POS + C@ R> R>  WORST WORST
   DIR-CONTENTS C! ;

: POS-TEXT  RED 2 0 TEXTC  -200 -20 MOVABS  ;
: BELL    7 EMIT ;
: NT   0 0  TEXTC ;
: W."   COMPILE POS-TEXT COMPILE vJot" 34 STRING
   COMPILE NT ; IMMEDIATE
: CLEAR-MSG  W."  " BLACK  1 P[illegible]FIL 190 15 RECREL ;
```

```
384 LIST

    ( MAPPER  PS constants & variables )           ( CJM*8/09/85)

  : CUR>INC    ( -xyz)  ZDIR @ NEWZ @ + H_?LEGAL NEWZ !
    XDIR @ NEWX @ +   YDIR @ NEWY @ +   XY_?LEGAL NEWY !
    DUP NEWX ! NEWY @ NEWZ @ ;
  : CUR>XYZ ( n/n--x/y/z)
         DUP 2 = IF DROP -1 THEN 6 = SWAP
         DUP 8 = IF DROP 0 -1  ELSE DUP 4 = IF DROP -1 0  ELSE
         DUP 2 = IF DROP 1  0  ELSE DUP 1 = IF DROP  0 1  ELSE
                    DROP 0 0  THEN THEN THEN THEN ROT ;
  : READ-JOYSTICK-CMD    ( -n)  PAUSE  READ-JOYSTICK OVER 16 <
    IF CUR>XYZ ELSE 2DROP 1 CELL-EXIT ! 0 0 0 THEN
    ZDIR ! OVER OVER YDIR ! XDIR ! OR ZDIR @ OR PAUSE ;
```

```
385 LIST

    ( Obstacle avoidance rules )                   ( CJM*8/09/85)
 *RESET-PS*    ( clear PS rule table )

 RULE: START-UP   PRIORITY: 10    ( initialize cell map )
       *IF*     NOT( STARTED @ )
       *THEN*   CELL-MAP-INIT
                STARTED TRUE   NEWDIR FALSE  CELL-EXIT FALSE
       *END*

 RULE: GET-DIR    PRIORITY: 0    ( get new direction request )
       *IF*   NOT( NEWDIR @ )
       *THEN* CPAD-CHECK
              READ-OVERIDE       IF CLEAR-MSG W."  OVER-RIDE" THEN
              READ-JOYSTICK-CMD IF CLEAR-MSG SCAN-DIR
                                   NEWDIR TRUE  THEN
       *END*
```

```
386 LIST

    ( Direction checking rules )
 RULE: RED-OBJ   PRIORITY: 1    ( destination obstructed )
       *IF*  NEWDIR @  DIR-CONTENTS C@ HIGH-OBS =
       *THEN* OVERIDE @
                 IF DIR-CONTENTS FALSE
                 ELSE NEWDIR FALSE W." OBSTRUCTED PATH" BELL THEN
       *END*
 RULE: YELLOW-OBJ  PRIORITY: 1    ( destination hazardous )
       *IF*  NEWDIR @  DIR-CONTENTS C@ LOW-OBS =
       *THEN*  W." HAZARDOUS AREA" BELL
               DIR-CONTENTS FALSE
       *END*
 RULE: WHITE-OBJ  PRIORITY: 1    ( destination blocked by wall )
       *IF*  NEWDIR @  DIR-CONTENTS C@ WALL =
       *THEN*  W." BLOCKED BY WALL" BELL  NEWDIR FALSE
       *END*
```

```
387 LIST

    ( Move ok rule )                                ( CJM*8/09/85)

 RULE: MAKE-MOVE    PRIORITY: 1    ( destination clear - make move)
       *IF*  NEWDIR @  NOT( DIR-CONTENTS C@ )
       *THEN*  CUR>INC  RUNNING  OVERIDE FALSE  NEWDIR FALSE
       *END* .

 RULE: EXIT-MAP    PRIORITY: 5    ( exit from cell map )
       *IF*  CELL-EXIT @
       *THEN*  CELL-MAP-EXIT  HALT
       *END*
```

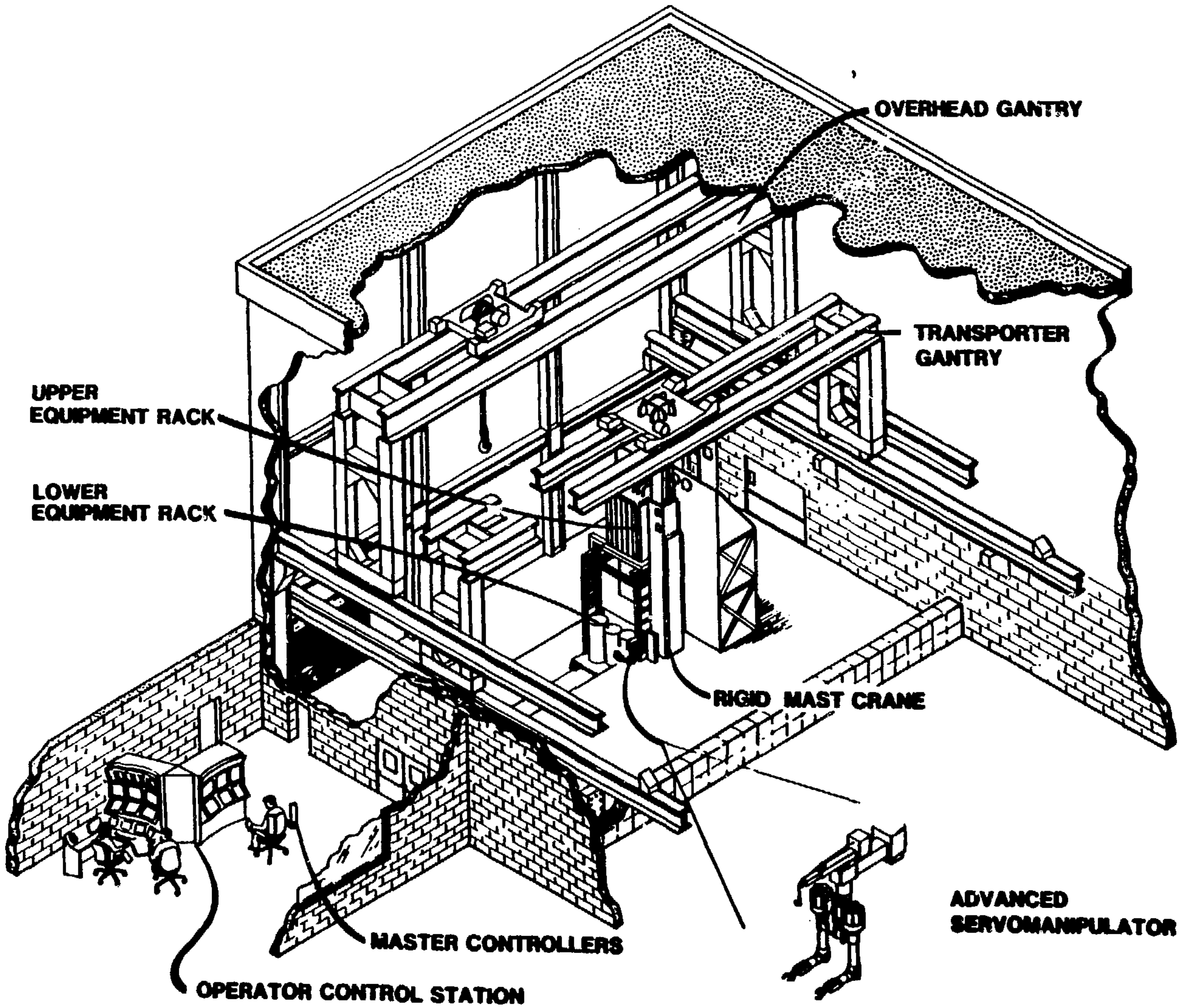
Fig. 1. Concept of the Advanced Integrated Maintenance System (AIMS).

Fig. 2. The Advanced Servomanipulator (ASM).

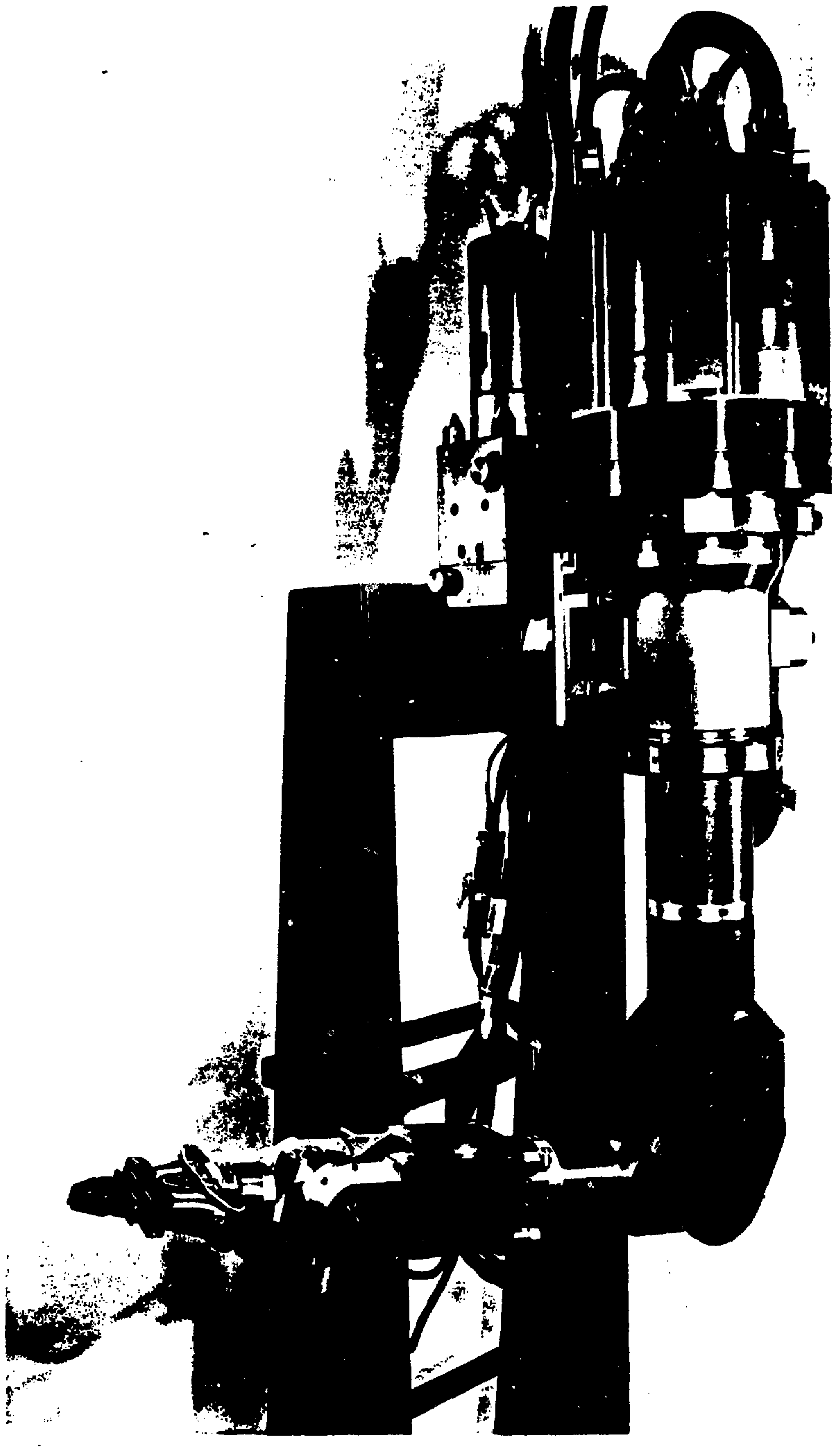
Fig. 3. Graphic cell map display.

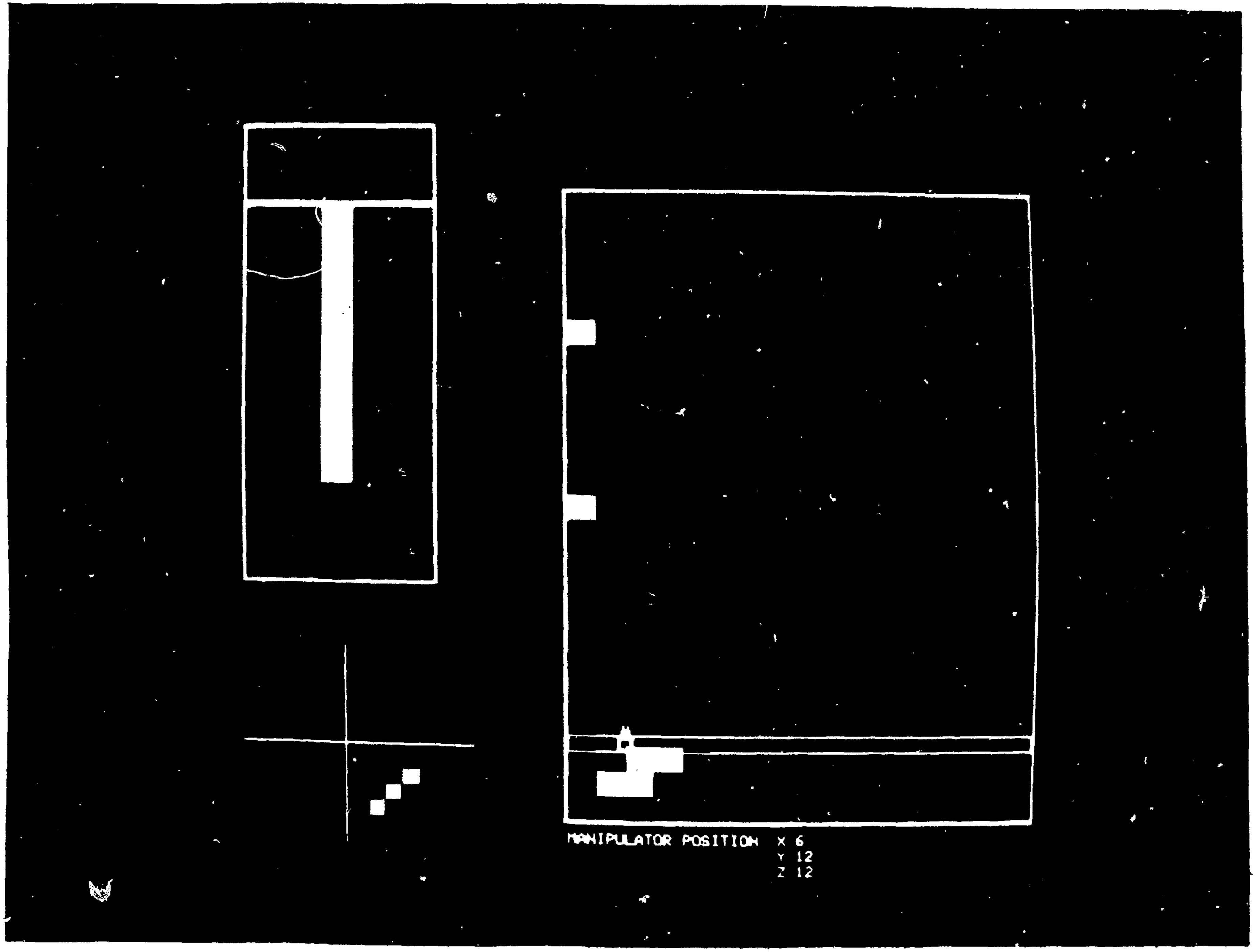
ORNL-DWG 85-6865R

THE ADVANCED INTEGRATED MAINTENANCE SYSTEM



ornl




MANIPULATOR POSITION X 6
Y 12
Z 12