two entry points are.

The items are:

```
ENT DATA STEP;                              A PROC VARIABLE CONTAINING THE
          PROC() ENTRY:=SMTJOB;               TASK ENTRY POINT ADDRESS
          STACK STKENTRY:=TASKSTK; A STACK VARIABLE CONTAINING THE
ENDDATA;                                      STACK ADDRESS OF RO
```

C3.4 DATA TEP NEW DSMTB1

This data brick is used at initialization time. The 2 items are temporary storage places of the stack and task entry points at stack initialization time.

The proc GETSTEP puts the 2 pieces of data into the data brick and the proc USERTASKINIT uses this when calling STKINIT.

The items are:

PROG () ENTRY	A proc variable to temporary stores the SMTJOB address of the task in question
STACK STKENTRY	A stack variable to temporary stores the stack address of the task in question

C3.5 DATA TASKDATA - ADDITION DSMTB3

This data brick contains the dynamic data of the operating system at any time of its operation.

A further array called BADLOAD was included to reflect the successfullness of a load.

The array of byte is initialized with a 4 in them. If the task exists and is loaded successfully the byte is made = 0. If the task exists and is not loaded successfully the byte is made = 1. These last 2 operations are carried out by the LOADER program.

The item is:

ARRAY (NUSERTASKS) BYTE BADLOAD = 4

C3.6 DATA TU58LDR NEW DSMTB3

This data brick contains two flags. The LDRFLAG1 is used to check whether a TU58 operation was successful. If a loader error occuss LDRFLAG1 is not equal to 0. This flag is used by the loader proc.

The second flag is LOADEDFLAG. This flag distinguishes to the STARTUP task whether it is the first time the STARTUP task is running. When the system is loaded for the first line LOADEDFLAG is zero. After all the user tasks have been loaded into memory LOADEDFLAG is made = 1. If we now restart the processor by saying 0 G the tasks will not be reloaded since the flag is non zero. The flag is used by SETFMQ.

The items are:

INT LDRFLAG1	Error flag in loading a task
BYTE LOADEDFLAG	First time round flag

C3.7 DATA TUDRIVER2 NEW DSMTB2

The data brick contains the data required for the loading of a task using the TU58 tape loader proc WORKTU.

The data brick contains a record defined by the mode.

MODE IOPRT (BYTE FCN, UNIT, INT BCOUNT, PBLK REF BYTE BUF1)

BYTE FCN	The function required of the TU58: 2 = READ 3 = WRITE
UNIT	Required unit No: 0 used all the time cannot be operator selected at present.
INT BCOUNT	Number of bytes to transfer (must be even).
PBLK	Starting physical block on tape (0 to 511).
REF BYTE BUF1	Pointer to first byte in buffer.

The items in the brick are:

IOPKT LD1OP1	The record of the above mentioned mode.
ARRAY (512)BYTE LDBUF	The buffer used to store the data transfered from the tape before being relocated. The size 512 chosen to equal 1 block of tape.
INT PARWINDOW	The first PAR constant of the task to be loaded.

PART D

General Topics

D1 Keyboard Commands

Certain functions are available to the operator via the CTLA function. the operator may communicate to the operating system and do such commands as STOPPING and STARTING of tasks etc. This function is still available to the operator.

The CTLA task now appears as a user task and is not a system task. The one limitation of this, is that the operator cannot look at memory locations (using the LK command) other than those in the common area.

Certain other functions installed in the CTLA task namely:

D1.1 LT

Entering LT starts the loader task TU58. This enables the operator to load a task from cassette on line.

A request is made for the operator to enter the task number to be loaded.

The cassette should be installed in drive 0.

D1.2 FA

Entering FA will list the facilities in use. The facility number is printed followed by the task using the facilities.

D1.3 EV

Entering EV will list the status of all the tasks.

The task number is printed out first followed by either:

D	-	task delayed.
S	-	task stopped.
WFn	-	task awaiting facility n.
WEn	-	task awaiting event n.

Features FA and EV have been added to assist in debugging systems during the development stage.

PART E

The Development of a User System

What does system generation involve

The building of a system to operate under the new SMT can be split into 2 areas.

1. The defining of the common area.
2. The defining of the tasks.

E1 Defining the Common Area

E1.1 Introduction

The common area will contain all or part of the following item:

1. The o/s.
2. All user defined device drivers.
3. The system database.
4. The common data bricks (Bricks used by 2 or more tasks).
5. The common proc bricks (Procs used by 2 or more tasks).

Good multitasking principles would require that the common data bricks should be kept to a minimum at all times keeping the tasks as independent as possible.

Once the common area has been defined the common area size should be estimated since the larger the common area is made the smaller the task area will become.

E.g.1 Common area take PAR0 - 2 i.e. 12k
Task area will be PAR3 - PAR6 16k

E.g.2 Common area PAR0 - 3 i.e. 16k
Task area PAR4 - 6 i.e. 12k

The operating system is approximately 7500 decimal words long.

This common area should not have any EXT references defined, i.e. all the cross references should be defined in the common area.

If the common area is to be enlarged Appendix 6 outlines all the areas of change.

Once the common area has been estimated as 12k or 16k the user tasks images may be built.

E1.2 Editing

E1.2.1 O/S Modules

If a change is not made to the common area size, then the o/s module apart from 2 need not be edited. The two that are edited is DSMTU2 which will contain all the user task specification and also DSMTU1 which will contain any additional communication drivers. If the common area is to be enlarged, Appendix 6 outlines all the areas of change.

PART E

The Development of a User System

What does system generation involve

The building of a system to operate under the new SMT can be split into 2 areas.

1. The defining of the common area.
2. The defining of the tasks.

E1 Defining the Common Area

E1.1 Introduction

The common area will contain all or part of the following item:

1. The o/s.
2. All user defined device drivers.
3. The system database.
4. The common data bricks (Bricks used by 2 or more tasks).
5. The common proc bricks (Procs used by 2 or more tasks).

Good multitasking principles would require that the common data bricks should be kept to a minimum at all times keeping the tasks as independent as possible.

Once the common area has been defined the common area size should be estimated since the larger the common area is made the smaller the task area will become.

E.g.1 Common area take PAR0 - 2 i.e. 12k
Task area will be PAR3 - PAR6 16k

E.g.2 Common area PAR0 - 3 i.e. 16k
Task area PAR4 - 6 i.e. 12k

The operating system is approximately 7500 decimal words long.

This common area should not have any EXT references defined, i.e. all the cross references should be defined in the common area.

If the common area is to be enlarged Appendix 6 outlines all the areas of change.

Once the common area has been estimated as 12k or 16k the user tasks images may be built.

E1.2 Editing

E1.2.1 O/S Modules

If a change is not made to the common area size, then the o/s module apart from 2 need not be edited. The two that are edited is DSMTU2 which will contain all the user task specification and also DSMTU1 which will contain any additional communication drivers. If the common area is to be enlarged, Appendix 6 outlines all the areas of change.

E1.2.2 Common Area Modules other than the o/s Modules

These modules are written as standard RTL/2 modules i.e. the specification of the OPTIONS, TITLES, EXT DATA BRICKS etc are defined as a normal module.

The three should be segregated, i.e. database, common data brick module and common proc module, to facilitate clarity.

E1.3 Set up DSMTU2

E1.3.1 General

This module contains all the user task data i.e. it contains the user task priority, status, where located on tape and PAR settings.

When all the task data has been put in the module, the module should be compiled and a check made that it has a length equal to octal 1000 or decimal 512. The reason for this value is that this module is loaded into the machine as 1 block of tape whenever a task is loaded on line. The load is started at location octal 400. (See Figure E1)

This module replaces the data brick. TKDFN which previously held all the user task data.

E1.3.2 PAR Assignement and Tape layout

When laying out the SMT system manually, two additional areas have to be looked at and calculated. Namely the PAR constants to be assigned to the tasks and the layout of the tasks on the tape.

At the on-set of the project a table should be constructed of all the proposed tasks. This table should list all the tasks and the lengths of the tasks in decimal bytes as produced by the task builder map. (see TABLE 1)

To this figure should be added spare memory space in the event that the task is modified and the length exceeds the present length. Depending on the type of task spare memory should be allocated.

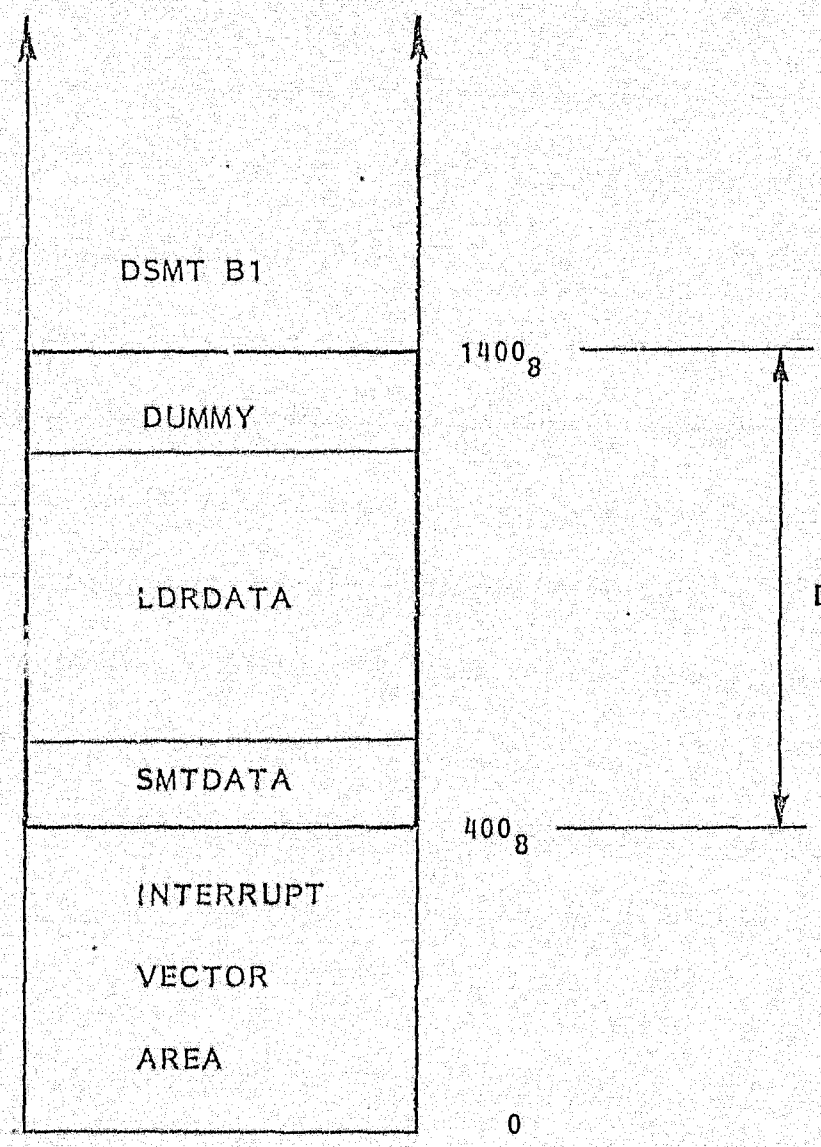
This new figure i.e. (actual length plus spare) should be rounded off to the nearest 512 bytes. The table in Appendix 4 should be of use. To give an indication of how much spare memory space the task has the third column is filled in.

From the same table in Appendix 4 the number of blocks required to fit this memory space of the individual task can be calculated and filled in the fourth column. (e.g. the SMTGTLA task occupies 2560 bytes of memory which is equivalent to 5 blocks of tape).

The table will also give us the number of relocation constants, i.e. blocks, are required, this is filled in the sixth column.

Figure E1

MEMORY MAP DETAILING THE LOCATION OF DSMTU2



This table now fully specifies the task lengths etc. From this table the absolute blockno on tape and the actual PAR constants can be calculated as described below.

NOTE: If at any time during development the task length increases beyond that allocated the constants will have to be recalculated.

E1.3.2.1 PAR Assignments
(see Table 2 and APR layout sketches)

After having listed the number of PAR's required by the individual tasks, the actual PAR's are calculated.

The method used is to allocate the full amount used used by the common area, i.e. if the common area size is 12K words and only 10K words are used, PAR's 0-600 are allocated to cover the 12K words. In the example the common area is 16K and thus the PAR values range from 0 to 1000. The first task starts at 1000.

Therefore for a : 12K common area the first task
PAR constant starts at 600
: 16K common area the first task
PAR constant starts at 1000.

The tasks then follow accumulatively i.e. if task one required 50 PAR constants the second task will be at 650 (for 12K common area).

If a task is longer than 4K words, the second PAR register will also have to be loaded. A register may contain a maximum of 200 which is equivalent to 4K words.

E.g. in the example LMAZ5 requires 360 relocation constants. Therefore 200 are allocated to APR4 and 160 to APR5.

The APR's which are not used are allocated a dummy location.

E1.3.2.2 Tape Layout
(see Table 3 and tape layout sketches)

Block 0 contains the startup/absolute loader program. This program is the same for any system built under the new SMT.

Block 1 contains the data brick DSMTU2 built as a single image. The image occupies 1000 octal bytes of memory. (1 block on tape.)

Block 2 is spare.

Block 3 contains the common area. The length of the common data is either:
48 blocks for a 12k common area

TABLE I

APR AND TAPE DOCUMENTATION / SELECTION

This module contains all the information used to set up the MAAIS system to operate on the new SMT operating system. The data contained in this module is used to set up the module DSMTU2 which contains the user task data as well as the info required to load the task images from the development machine to the tape.

C.CHARALAMBOUS 17-AUGUST-83

Task Number and Name	Module Actual	Task Length (decimal bytes) Allowed	Extra	No of Tape blocks read (dec)	(oct)	No of Relocation constants read (octal)
1. INSTRUCT	SMTCTLA	1956	2560	604	5	5	50
2. PRECH	MAZ1	3266	4096	830	8	10	100
3. PALRM	MAZ3	922	1536	614	3	3	30
4. MOVDIS	MAZ4	1880	2560	680	5	5	50
5. OPERATOR	LMAZ5	12566	15360	2794	30	36	360
6. ECHO	MAZ16	196	512	316	1	1	10
7. WTCHDOG	LMAZ7	3504	4096	594	8	10	100
8. ENGFUNC	MAZ8	2372	3072	362	6	6	60
9. REPORTS	LMAZ9	2710	3584	874	7	7	70
10.RINGALRM	MAZ10	226	512	286	1	1	10
11.STARTUP	LMAZ11	522	1024	502	2	2	20
OPERATING SYSTEM AND COMMON AREA:		-	16384	-	64	100	1 00

APR ASSIGNMENT

OPERATING SYSTEM AND COMMON AREA:	APR 0	OCTAL 000
	APR 1	OCTAL 200
	APR 2	OCTAL 400
	APR 3	OCTAL 600
I/O AREA	APR 7	OCTAL 7600

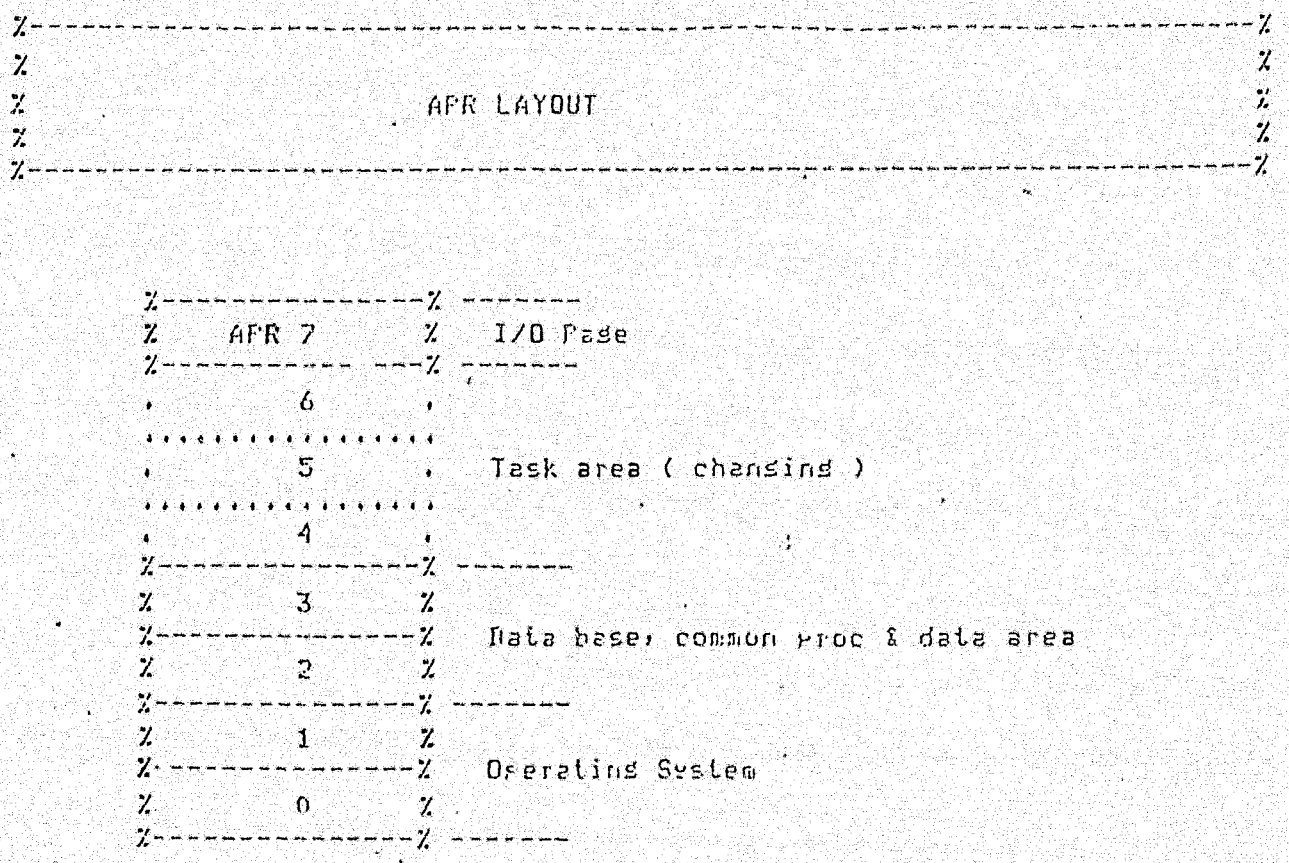


TABLE 2

Task No & Name	Module	APR's req'd	APR 4	APR 5	APR6
1. INSTRUCT	SMTCTLA	50	1000	5000	5000
2. PRECM	MAZ1	100	1050	5000	5000
3. PALRM	MAZ3	30	1150	5000	5000
4. WOVDIS	MAZ4	50	1200	5000	5000
5. OPERATOR	LMAZ5	360	1250	1450	5000
6. ECHO	MAZ16	10	1630	5000	5000
7. WTCHDOG	LMAZ7	100	1640	5000	5000
8. ENGFUNC	MAZ8	60	1740	5000	5000
9. REPORTS	LMAZ9	70	2020	5000	5000
10. RINGALRM	MAZ10	10	2110	5000	5000
11. STARTUP	LMAZ11	20	2120	5000	5000

NOTE: The APR value 5000 was used as the dummy APR number. At this location in MAAIS there is no memory (physical address 500 000). The highest constant we can use is 7777 which maps into the I/O page. A value higher than 7777 cannot be used as the dummy constant since the M/M unit strips off the bits 13-16 of the relocation constant, ie the constant 10 000 would appear to the M/M unit as 0 000 ... pointing to the operating system.... very....very....dangerous!

PHYSICAL MEMORY:

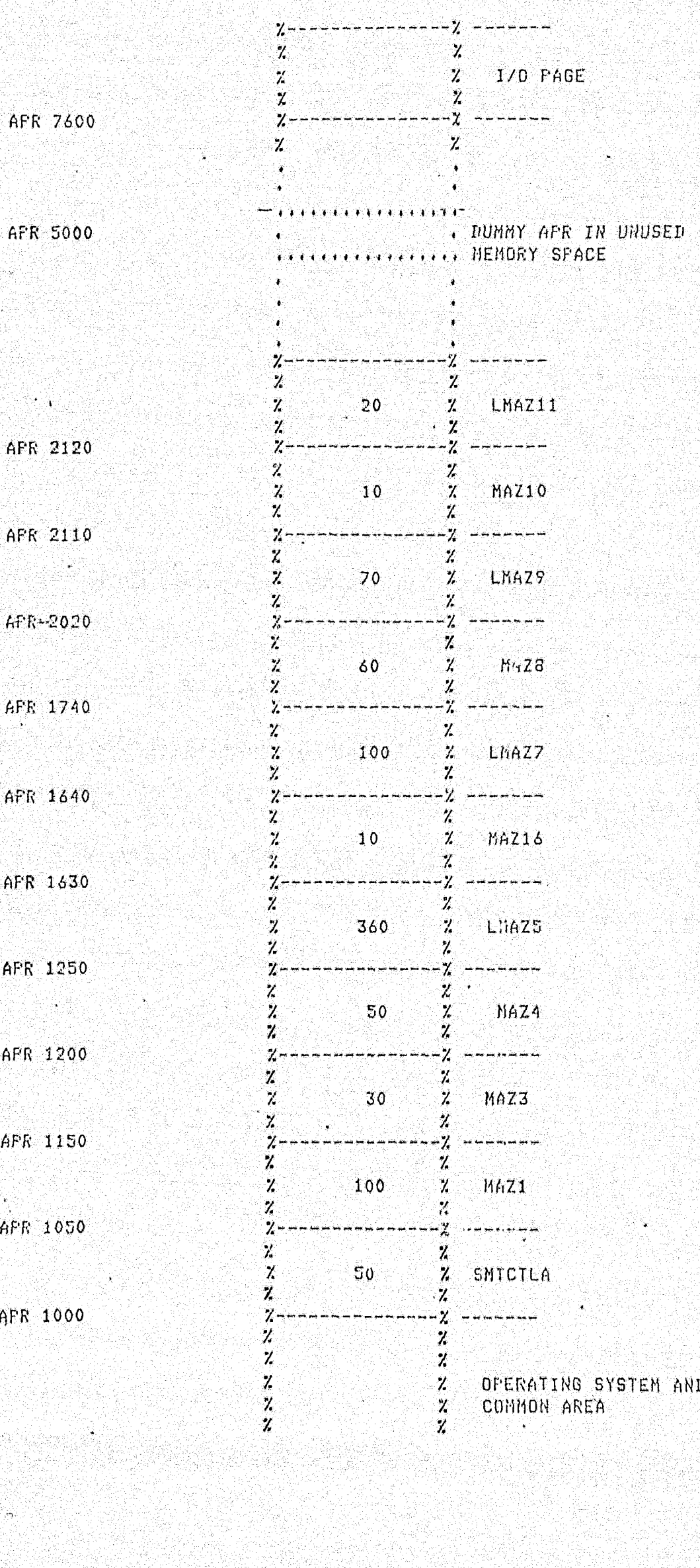


TABLE 3

TAPE ASSIGNMENT

Task No and Name	Module rea'd	Blocks (decimal)	Actual block Number (decimal)	Length (Octal)	Tape (octal)
1. INSTRUCT	SMTCTLA	5	67	103	5 000
2. PRECM	MAZ1	8	72	110	10 000
3. PALRM	MAZ3	3	80	120	3 000
4. WOVDIS	MAZ4	5	83	123	5 000
5. OPERATOR	LMAZ5	30	88	130	36 000
6. ECHO	MAZ16	1	118	166	1 000
7. WTCHDOG	LMAZ7	8	119	167	10 000
8. ENGFUNC	MAZ8	6	127	177	6 000
9. REPORTS	LMAZ9	7	133	205	7 000
10. RINGALRM	MAZ10	1	140	214	1 000
11. STARTUP	LMAZ11	2	141	215	2 000

!
!
!- used when loading onto tape
from RSX dev syst

TAPE LAYOUT:

OCTAL	DECIMAL		
141	215	2	LMAZ11
140	214	1	MAZ10
133	205	7	LMAZ9
127	177	6	MAZ8
119	167	8	LMAZ7
118	166	1	MAZ16
88	130	30	LMAZ5
83	123	5	MAZ4
80	120	3	MAZ3
72	110	8	MAZ1
67	103	5	SMTCTLA
		64	OPERATING SYSTEM AND COMMON AREA
3	3	1	SPARE
2	2	1	LOADER DATA
1	1	1	STARTUP AND RELOCATION PROGRAM
0	0		

64 blocks for a 16k common area

Block X1 CTLA task. - User Task 1

Block X2 User written task - User Task 2

Block Xn User written task - User Task n

The calculation of the number of blocks is given in Appendix 4.

The user tasks should all be rounded to a block number (i.e. 512 byte boundary). Spare tape blocks should be left after each task so that when modifications are made the module DSMTU2 need not be recompiled and task built.

Two points should be remembered:

1. Each tape block is of length 512 byte i.e. octal 1000. The length of each task should be looked at and it is at the designers discretion how much tape space should be left as spare. Tasks that are likely to be modified often should be allocated a fair amount of space and vice versa.

2. The total number of blocks on one tape is 512.

A table exists in Appendix 4 which lists for a given task length in octal bytes, how many tape blocks are required.

An example of an approach used for a given system is shown overleaf.

E1.4 Compiling and Common Area Modules

See Reference 1.

The following should be compiled:

1. All common area modules.
2. All o/s modules.

E1.4.1 Compiling DSMTU2

When compiling DSMTU2 after having inserted all the user task data in the module, it is imperative that, the macro length should be decimal 512 or octal 1000. This module length can be altered by changing the array length of the data brick called DUMMY.

E1.5 Modifying the common Area

If a common area module is modified, the module is edited, compiled and the common area task built. All the user task have to be task built again.

The common area and user tasks are reloaded onto the tape and the tape run into the target machine.

64 blocks for a 16k common area

Block X1 CTLA task. - User Task 1

Block X2 User written task - User Task 2

Block Xn User written task - User Task n

The calculation of the number of blocks is given in Appendix 4.

The user tasks should all be rounded to a block number (i.e. 512 byte boundary). Spare tape blocks should be left after each task so that when modifications are made the module DSMTU2 need not be recompiled and task built.

Two points should be remembered:

1. Each tape block is of length 512 byte i.e. octal 1000. The length of each task should be looked at and it is at the designers discretion how much tape space should be left as spare. Tasks that are likely to be modified often should be allocated a fair amount of space and vice versa.

2. The total number of blocks on one tape is 512.

A table exists in Appendix 4 which lists for a given task length in octal bytes, how many tape blocks are required.

An example of an approach used for a given system is shown overleaf.

E1.4 Compiling and Common Area Modules

See Reference 1.

The following should be compiled:

1. All common area modules.
2. All o/s modules.

E1.4.1 Compiling DSMTU2

When compiling DSMTU2 after having inserted all the user task data in the module, it is imperative that, the macro length should be decimal 512 or octal 1000. This module length can be altered by changing the array length of the data brick called DUMMY.

E1.5 Modifying the common Area

If a common area module is modified, the module is edited, compiled and the common area task built. All the user task have to be task built again.

The common area and user tasks are reloaded onto the tape and the tape run into the target machine.

If the priority of the task just loaded has been modified the system will not install it into the o/s array, what has to be done is interrupt the processor and initiate a OG. The reason for this is that the STATAD array is not set up again after a reload of a task only. By doing a OG the STATAD array is set up with the data contained in DSMTU2 which would have been altered by the reload.

E2 Defining the User Tasks

E2.1 Editing

Each task should be programed into one module. An example of the recommended layout of the task is shown overleaf.

Each task has to have 2 elements defined for the task to link to the operating system.

The stack for all tasks is declared in the same manner and is declared as:

```
STACK TASKSTK 300
```

The task base proc should be placed in the proc called SMTJOB

```
e.g.  ENT PROC SMTJOB
          PROC RINGALRM ()
      ENDPROC
```

where RINGALRM is the task base proc.

When task building a task, the TKB puts the address of these 2 declarations into the module SMTTE. The o/s links dynamically to these 2 locations at run time.

Two examples of tasks have been shown on the following pages:

E3 Task Building

The development work was carried out under RSX 11M. These notes refer to the building of SMT systems under RSX 11M.

When task building 4 things are important:

1. The common area should be task built first using TKB. The common area is built with a .STB file which will contain all the global cross references required by the user tasks.

   A typical indirect command file is shown in Figure E2.

2. After having completed the building of the common area, the user tasks images are built one at a time using the global cross reference file produced in step (1). This file satisfies all the EXT declarations in the user modules.

   A typical indirect command file to build a task is shown in Figure E3.

   Another indirect command file is shown, which will build all the tasks by a call of the indirect command file of described in 2 is shown in Figure E4.

3. An image called LDRDATA containing the user task data should be built and loaded into block 1 of the tape. This is used at load time, since it contains all the user task data. If at any stage the data contained in DSMTU2 is changed, this image should be rebuilt and reloaded into block 1 of the tape. (see Figure E5)

4. An image called SARP containing the bootstrap startup program should be built and loaded into block 0 of the tape. Having been loaded once on the tape it need not be altered again. (see Figure E6)

   An example of the .MAP produced by the SMTTKB command file is also shown in Figure E7.

E4 Bootstrap Procedure and User Task Data Brick

To Bootstrap the system off the tape into the computer memory, two additional features have been added to the system. Namely a bootstrap/startup programe has been put in block 0 of the tape which controls the loading in of the user task data, and the common area.

This startup program called SARP is supplied in source form and should be compiled and task built using the TKB command file shown overleaf.

This startup image need never be reloaded into block 0 of the tape after the first time. The program is totally independent of the system being designed. The program loads in the user data brick, residing in block 1 and reads from the data brick where the starting block number of the common area is on tape and also the length of the common area to be read in.

Therefore the DSMTU2 module needs to be compiled and task built using the LDRDATA command file. this is loaded into block 1 of the tape.

NOTE: If any variables are changed in DSMTU2, the common area is task built, as well as the DSMTU2 module. This latter module must be loaded into block 1.

E5 Loading of Images onto Tape

When loading images onto tape under control of RSX-11M the task TU58 should be in the users account.

By entering:

RUN TU58

The TU58 loader task is activated and by answering the questions the images are loaded one at a time.

At present this task cannot be controlled by an indirect command file. This feature is being looked at. This implies that all the tasks have to be loaded one at a time under the programmers control.

An example is given of the procedure used to load all the images onto tape on the next page.

The startup and relocation program only needs to be loaded once on the tape.

The starting block numbers also appear in the tape and PAR assignment

```
%
%                                    MAZ16                                    %
%                                                                             %
%                          TASK 6 - ECHO CHARACTER TO VDU                     %
%                                                                             %
%-----------------------------------------------------------------------------%

TITLE
        MAZ16 - ECHO CHARACTER TO VDU

%*****************************************************************************%

OPTION (1) EI,SL,BC,BS;

%*****************************************************************************%

%-----------------------------------------------------------------------------%
%        DATA BRICKS DEFINED IN THIS MODULE                                   %
%                  DATA ISCECHODATA                                           %
%        PROCS DEFINED IN THIS MODULE                                         %
%                  ENT PROC ECHO                                              %
%                                                                             %
%-----------------------------------------------------------------------------%

LET EVECHO =   9;                                % ECHO TO ISC               %
LET FAISC  =  13;                                % FACILITY - ISC VDU        %


%-----------------------------------------------------------------------------%
%                  EXTERNAL PROCS                                             %
%-----------------------------------------------------------------------------%

EXT PROC (INT) WAIT, SECURE, RELEASE;
EXT PROC (BYTE) OUTISC;

%-----------------------------------------------------------------------------%
%                  EXTERNAL DATA BRICKS                                       %
%-----------------------------------------------------------------------------%

%-----------------------------------------------------------------------------%
%                  SVC DATA BRICKS                                            %
%-----------------------------------------------------------------------------%

SVC DATA RRERR ; LABEL ERL; INT ERN; PROC(INT)ERP; ENDDATA;
SVC DATA RRSIO ; PROC ()BYTE IN; PROC (BYTE)OUT; ENDDATA;
SVC DATA RRSED ; BYTE TERMCH, IOFLAG;ENDDATA;
SVC DATA STKUSG; INT USAGE, LINE ; ENDDATA;

%-----------------------------------------------------------------------------%
%                  ENT DATA BRICKS                                            %
%-----------------------------------------------------------------------------%

DATA ISCECHODATA;
        BYTE ECHOCHAR;
ENDDATA;

%-----------------------------------------------------------------------------%
%                  ENT STACK                                                  %
%-----------------------------------------------------------------------------%

ENT STACK TASKSTK %20;


%-----------------------------------------------------------------------------%
%                  TASK ENTRY POINT                                           %
%-----------------------------------------------------------------------------%

ENT PROC SMTJOB();
        ECHO();
```

```
%-------------------------------------------------------------------------%
%                    ENT STACK                                            %
%-------------------------------------------------------------------------%

ENT STACK TASKSTK 120;


%-------------------------------------------------------------------------%
%                    TASK ENTRY POINT                                     %
%-------------------------------------------------------------------------%

ENT PROC SMTJOB();
        ECHO();
ENDPROC;


%-----------------------------------------------------------PROC ECHO-------%
%                                                                         %
% TO PERMIT THE VDU OVERVIEW DISPLAY TO BE UPDATED WHILE KEYBOARD INPUT   %
% IS IN PROGRESS. THIS MUST BE DONE AS A SEPERATE TASK AS FAISC CANNOT    %
% BE SECURED WHILE IN INISC (IT IS AN H-TASK)                             %
%-------------------------------------------------------------------------%

ENT PROC ECHO ();                       % TASK 6 - ECHO CHARACTER TO VDU %
ECHO1:
   WAIT(EVECHO);
   SECURE(FAISC);
       OUTISC(27);OUTISC(27);      % RESET TO VISIBLE CURSOR %
       OUTISC(ECHOCHAR);
   RELEASE(FAISC);
   GOTO ECHO1;

ENDPROC;
```

```
%-------------------------------------------------------------------------------%
%                                                                               %
%                                    MAZ 10                                     %
%                                                                               %
%                              TASK 10 - RING ALARM                             %
%                                                                               %
%-------------------------------------------------------------------------------%

TITLE
       MAZ10 - RING ALARM PROCEDURE

RING ALARM TASK

%-------------------------------------------------------------------------------%
% REV 1.1 10/6/80   A.D.HEHER                                                   %
%     1.2 23/7/80 ADH  ALARM COUNT AND RING (TASK 10)                           %
%     1.3 8/8/80  ADH  COMPRESSED ANODE ALARM FORMAT                            %
%     1.4 8/9/80  ADH  ALARMS LOST MESSAGE ON END OF LINE                       %
%-------------------------------------------------------------------------------%

%*********************************************************************************%

0   N(1) EI, SL;

%*********************************************************************************%

%-------------------------------------------------------------------------------%
%        DATA BRICKS DEFINED IN THIS MODULE                                     %
%                                                                               %
%        INDEX TO PROCEDURES                                                    %
%            RINGALRM  - RING ALARM    ** TASK 10 **                            %
%-------------------------------------------------------------------------------%

LET RAB = REF ARRAY BYTE;

LET EVSALRM=11;                 %          SOUND ALARM                          %
LET EOM     = 3;

%-------------------------------------------------------------------------------%
%        EXTERNAL PROCS                                                         %
%-------------------------------------------------------------------------------%

EXT PROC(INT)       WAIT,DELAY;
EXT PROC(BYTE) OUTTTY;


%-------------------------------------------------------------------------------%
%        EXTERNAL DATA                                                          %
%-------------------------------------------------------------------------------%

EXT DATA ALRMDATA;                                  % AUDIBLE ALARM DATA      %
   INT ALRMN,                                % ALARM  NO. = PRIORITY  (1-HI)   %
       ALRMON;                                     % ALARM ON/OFF FLAG         %
   ARRAY (5) INT NRINGS;                     % NO. OF RINGS ( BELLS ) OUT       %
   ARRAY (5) INT TBRINGS;                  % TIME BETWEEN RINGS,CLOCK TICKS     %
   ARRAY(6) BYTE ALRMP;
ENDDATA;

%-------------------------------------------------------------------------------%
%        SVC DATA BRICKS                                                        %
%-------------------------------------------------------------------------------%

SVC DATA RRERR; LABEL ERL; INT ERN; PROC(INT) ERP ENDDATA;
SVC DATA RRSIO; PROC () BYTE IN; PROC (BYTE) OUT ENDDATA;
SVC DATA RRSED; BYTE TERMCH,IOFLAG ENDDATA;

%-------------------------------------------------------------------------------%
                                                                                %
```

```
%------------------------------------------------------------------------------%
%                                                                              %
%                                    MAZ 10                                    %
%                                                                              %
%                            TASK 10 - RING ALARM                              %
%                                                                              %
%------------------------------------------------------------------------------%

TITLE
       MAZ10 - RING ALARM PROCEDURE

RING ALARM TASK

%------------------------------------------------------------------------------%
% REV  1.1  10/6/80   A.D.HEHER                                                %
%      1.2  23/7/80   ADH   ALARM COUNT AND RING (TASK 10)                     %
%      1.3  8/8/80    ADH   COMPRESSED ANODE ALARM FORMAT                      %
%      1.4  8/9/80    ADH   ALARMS LOST MESSAGE ON END OF LINE                 %
%------------------------------------------------------------------------------%

(*****************************************************************************%

OPTION(1) EI, SL;

(*****************************************************************************%

%------------------------------------------------------------------------------%
%          DATA BRICKS DEFINED IN THIS MODULE                                  %
%                                                                              %
%          INDEX TO PROCEDURES                                                 %
%              RINGALRM  - RING ALARM    ** TASK 10 **                         %
%------------------------------------------------------------------------------%

LET RGB   REF ARRAY BYTE;

LET EVSALRM=11;                 %          SOUND ALARM                         %
LET EOM    = 3;

%------------------------------------------------------------------------------%
%          EXTERNAL PROCS                                                      %
%------------------------------------------------------------------------------%

EXT PROC(INT)       WAIT,DELAY;
EXT PROC(BYTE) OUTTTY;


%------------------------------------------------------------------------------%
%          EXTERNAL DATA                                                       %
%------------------------------------------------------------------------------%

EXT DATA ALRMDATA;                                  % AUDIBLE ALARM DATA      %
    INT ALRMN,                                % ALARM  NO. = PRIORITY  (1-HI) %
        ALRMON;                                     % ALARM ON/OFF FLAG       %
    ARRAY (5) INT NRINGS;                    % NO. OF RINGS ( BELLS ) OUT      %
    ARRAY (5) INT TBRINGS;                  % TIME BETWEEN RINGS,CLOCK TICKS  %
    ARRAY(6) BYTE ALRMP;
ENDDATA;

%------------------------------------------------------------------------------%
%          SVC DATA BRICKS                                                     %
%------------------------------------------------------------------------------%

SVC DATA RRERR; LABEL ERL; INT ERN; PROC(INT) ERP ENDDATA;
SVC DATA RRSIO; PROC () BYTE IN; PROC (BYTE) OUT ENDDATA;
SVC DATA RRSED; BYTE TERMCH,IOFLAG ENDDATA;

%------------------------------------------------------------------------------%
```

```
SVC DATA RRERR; LABEL ERL; INT ERN; PROC(INT) ERP ENDDATA;
SVC DATA RRSIO; PROC () BYTE IN; PROC (BYTE) OUT ENDDATA;
SVC DATA RRSED; BYTE TERMCH; IOFLAG ENDDATA;

%-------------------------------------------------------------------------%
%        ENT AND LOCAL DATA BRICKS                                        %
%-------------------------------------------------------------------------%


%-------------------------------------------------------------------------%
%                ENT STACK BRICK                                          %
%-------------------------------------------------------------------------%

ENT STACK TASKSTK 120;

%-------------------------------------------------------------------------%
%                TASK ENTRY POINT                                         %
%-------------------------------------------------------------------------%

ENT PROC SMTJOB();
        RINGALRM();
ENDPROC;

%-------------------------------------------------------------------------%
%                                                                         %
%                        TASK 10                                          %
%                        -------                                          %
%                                                                         %
%-------------------------------------------------------------------------%


ENT PROC RINGALRM ();

    ERL:=WAITAGAIN;
    OUT:=OUTTTY;
WAITAGAIN:
    WAIT(EVSALRM);
RINGAGAIN:
    IF ALRMON=0 THEN
                GOTO WAITAGAIN                          % ALARM OFF      ;
    END;

    TO NRINGS(ALRMN) DO
                OUT(7)                                              % RING ;
    REP;

    OUT(EOM);                                  % RELEASE TTY BETWEEN RINGS ;
    DELAY(TBRINGS(ALRMN));                          % TIME BETWEEN RINGS ;
    GOTO RINGAGAIN;

ENDPROC;
```

FIG. E2

SMT.CMD

```
.ENABLE QUIET
.ENABLE SUBSTITUTION
.SETS NUL ""

.OPEN DSMT4.TKB
.DATA DSMT4/-HD/SQ,DSMT4.MAP/SP/MA,DSMT4=
.DATA DL0:LOUCR
.DATA DL0:DSMTU2
.DATA DL0:DSMTB1
.DATA DL0:DSMTB2
.DATA DL0:DSMTB5
.DATA DL0:DSMTB3
.DATA DL0:RTLCTL
.DATA DL0:DSMTU1
.DATA DL0:TUDRV

.DATA DL0:MAZU2
.DATA DL0:LMAZD1
.DATA DL0:MAZCF
.DATA DL0:MAZCD

.DATA /
.DATA UNITS=0
.DATA STACK=0
.DATA PAR=DSMT4:0000:100000
.DATA //
.CLOSE

.OPEN DSMT4.RLV
.DATA ^LF!=DSMTB1,DSMTB2,DSMTB3,DSMTB5,DSMTU1,TUDRV,DSMTU2/XX
.DATA ^,MAZU2,LMAZD1,MAZCF,MAZCD/SS
.DATA 'NUL'
.CLOSE

RLV @DSMT4.RLV
TKB @DSMT4.TKB

PIP DSMT4.*/PU
```

FIG. E3

SMTTKB.CMD

```
.ENABLE QUIET
.ENABLE SUBSTITUTION
.ENA    GLOBAL
.SET. .OL ""

.SETS SMTASK P1
.SETS MAP P2
.SETS TEST P3

.IF TEST = 'Y' .GOTO 3

.4:
.ASKS SMTACK FILENAME TO BUILD ?
.TEST SMTASK
.IF <STRLEN> = 0 .STOP
.ASKS MAP MAP REQUIRED (P=PRINT, T=TERMINAL, ELSE NONE)?
.IF MAP <> 'P' .GOTO 1
.SETS MAP 'SY:LP.MAP/SP/MA'
.GOTO 3
.1:
.IF MAP <> 'T' .GOTO 2
.SETS MAP 'TI:'
.GOTO 3
.2:
.IF <STRLEN> = 0 .GOTO 3
;*** INVALID MAP CHOICE ***
.STOP
.3:
.OPEN 'SMTASK'.TKB
.DATA 'SMTASK'/-HD/SQ,'MAP','SMTASK'=
.DATA DL0:DSMT4.STB
.DATA DL0:SMTTE
.DATA 'SMTASK'
.DATA /
.DATA UNITS=0
.DATA STACK=0
.DATA PAR='SMTASK':100000:60000
.DATA //
.CLOSE


.OPEN 'SMTASK'.RLV
.DATA ,LP:=DSMTB1,DSMTB2,DSMTB3,DSMTB5,DSMTU1,DSMTU2,TUDRV/XX
.DATA ,MAZU2,LMAZD1,MAZCP,MAZCD,'SMTASK'/SS
.DATA 'NUL'
.CLOSE

RLV @'SMTASK'.RLV
TKB @'SMTASK'.TKB

PIP 'SMTASK'.*/PU

.IF TEST = 'Y' .GOTO 5

.GOTO 4

.5:
```

FIG. E4

MAZ.CMD

```
.ENABLE QUIET
.ENABLE SUBSTITUTION
.ENABLE GLOBAL

.SETS P2 'T'
.SETS P3 'Y'

@SMT

.SETS P1 'SMTCTLA'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'MAZ1'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'MAZ3'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'MAZ4'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'LMAZ5'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'MAZ16'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'LMAZ7'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'MAZ8'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'LMAZ9'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'MAZ10'
@SMTTKB 'P1' 'P2' 'P3'

.SETS P1 'LMAZ11'
@SMTTKB 'P1' 'P2' 'P3'
```

COMMON AREA - TASK BUILT

USER TASK 1 - TASK BUILT

ALL USER TASKS - TASK BUILT

ONE AT A TIME

FIG. E5

LDRDATA.CMD

```
.ENABLE QUIET
.ENABLE SUBSTITUTION

.OPEN LDRDATA.TKB
.DATA LDRDATA/-HD/SG,LDRDATA.MAP/SP/MA,LDRDATA=
.DATA DL0:DSMTU2

.DATA /
.DATA UNITS=0
.DATA STACK=0
.DATA PAR=LDRDATA:0000:1000
.DATA //
.CLOSE

TKB @LDRDATA.TKB

@SMT
```

FIG. E6

SARP.CMD

```
SARP/-HD/SQ,SARP,MAP/SP/MA,SARP=
DL0:SARP

/
UNITS=0
STACK=0
PAR=SARP:0000:1000
//
```

```
ARTITION NAME : SMTCTL
DENTIFICATION : 05AR83
ASK  UIC      : [220,1]
ASK ATTRIBUTES: -HD
OTAL ADDRESS WINDOWS: 1.
ASK  IMAGE  SIZE  : 992. WORDS
ASK ADDRESS LIMITS: 060000 063647
-W DISK BLK LIMITS: 000003 000002 000000 00000.

** ROOT SEGMENT: DSMT4

/W MEM  LIMITS: 060000 063647 003650 01960.
ISK BLK LIMITS: 000002 000005 000004 00004.
```

EMORY ALLOCATION SYNOPSIS:

SECTION				TITLE	IDENT	FILE
. BLK.1	(RW,I,LCL,REL,CON)	060000	000000 00000.			
SMTTE 1	(RW,I,LCL,REL,CON)	060000	000004 00004.			
		060000	000004 00004.	SMTTE	05AR83	SMTTE.OBJ;31
SMTCTL1	(RW,I,LCL,REL,CON)	060004	003644 01956.			
		060004	003644 01956.	.SMTCTL		SMTCTLA.OBJ;46

GLOBAL SYMBOLS:

CHANGE 003062	HUNLOC 002214	PTORTL 002662	RZZ 001400	R20 024162	SET 007412	TIMDAT 013460
CLEANU 011516	INSTDA 023156	PU 002342	R00 001400	R21 024174	SETDAT 017024	TIMEDA 023002
CLKDAT 023026	INSTK 020256	PUT2 013704	R01 023360	R22 024210	SETFMQ 004210	TKDFN 025464
CLKSTK 021052	INSTRU 060434-R	PUFDAT 023146	R02 023426	R23 002714	SETPAR 005616	TRACE 012776
CLOCK 003346	INTTY 026366	QEV 003222	R03 023440	R24 002662	SMTJOB 060414-R	TRACED 006156
CLOCK1 001670	IREAD 014010	QEVRRE 003316	R04 023456	R25 023514	SMT... 002226	TRAPDA 023132
CPUPER 006316	ITORB 002506	QR 002720	R05 023550	R26 024444	SPS 014662	TSECUR 012002
CREAD 017300	ITORF 002506	RBTDI 002506	R06 023554	R27 024436	START 006426	TSTSCR 011614
[illegible]	[illegible]	RECDAT 023030	R07 023627	R28 024606	STEP 060000-R	TWAIT 011722

FIG. E7
CTLA TASK MAP

table.

NOTE: The TU58 driver has to be INSTALLED but not MOUNTED under RSX.

E6 Loading tasks on Line

SMT has been enhanced so that the facility of loading individual tasks has been added.

If a task is to be modified the source program is modified and recompiled. This object code is then task built with the common area that resides in the m/c.

A flowchart of the steps to be followed when modifying a task is shown in Figure E8.

A check must be made that the task image length produced by the task builder is not longer than the memory space allocated to the task at system set up. If the new task image is larger than the space allocated, it will overwrite the next task in memory.

If this is the case the module DSMTU2 will have to be reconfigured and recompiled. DSMTU2 will have to be task built on its own and loaded into block 1 of the tape. The common area will have to be rebuilt with the modified DSMTU2.

The tasks are then reloaded onto tape at the new assignments and the tape run into the target m/c.

If the task image is smaller than memory space allocated then the process is easy. The new image is loaded onto tape and under the CTLA function, the task is loaded and initialized.

It is obvious that at the development stage, individual task length will have to be overestimated so that the long procedure of reassigning PAR values need not have to be done, every time a modification to the tasks are made.

E7 Initial Loading and Bootstrapping of DSMT

The method used to load the m/m SMT system has been modified to enable individual tasks to be loaded at run time. This has been made possible by task building each task individually and loading them into the target machines memory one at a time. These tasks are linked at run time as outlined previously.

Figure E9 and E10 indicates the procedure at startup.

The bootstrap on the BDV board runs in the first block (block 0) on tape which contains a startup procedure and an absolute loader called SARP. This proc is totally independent of the system being designed.

This startup proc then relocates itself up in memory so that it may not be overwritten by the common area which will be loaded at the lower location in memory.

The startup proc then loads in module DSMTU2 which contains all the user task data. The module also contains the length of the common area and where it resides a tape.

```
HEL CHRIS/NAZ

        RSX-11M BL26   MULTI-USER SYSTEM

GOOD MORNING
18-AUG-83 09:53 LOGGED ON TERMINAL TT0:

Logged onto the CIW 11/24 system.
>ALL DD:
>
>RUN TU58
TU58 IMAGE TRANSFER PROGRAM V1.2

Source File Name? SARP.TSK

File Transfer (F) or System Image Transfer (S)? S

In which unit is your cartridge mounted? 0

Starting Block Number on Tape (0-511)? 0

Number of Blocks Written = 1

Approx. Size of S.M.T. Image = 0K Bytes

Source File Name? LDRDATA.TSK

File Transfer (F) or System Image Transfer (S)? S

In which unit is your cartridge mounted? 0

Starting Block Number on Tape (0-511)? 1

Number of Blocks Written = 1

Approx. Size of S.M.T. Image = 0K Bytes

Source File Name? DSM14.TSK

File Transfer (F) or System Image Transfer (S)? S

In which unit is your cartridge mounted? 0

Starting Block Number on Tape (0-511)? 3
5  BLOCKS WRITTEN
10  BLOCKS WRITTEN
15  BLOCKS WRITTEN
20  BLOCKS WRITTEN
25  BLOCKS WRITTEN
30  BLOCKS WRITTEN
35  BLOCKS WRITTEN
40  BLOCKS WRITTEN
45  BLOCKS WRITTEN
50  BLOCKS WRITTEN
55  BLOCKS WRITTEN
60  BLOCKS WRITTEN

Number of Blocks Written = 61

Approx. Size of S.M.T. Image = 30K Bytes
```

STARTUP AND RELOCATION PROGRAM

TASK DATA BLOCK

COMMON AREA

```
Source File Name? SMTCTLA.TSK
File Transfer (F) or System Image Transfer (S)? S
In which unit is your cartridge mounted? 0
Starting Block Number on Tape (0-511)? 67
Number of Blocks Written = 4
Approx. Size of S.M.T. Image = 2K Bytes
```

USER TASK 1

```
Source File Name? MAZ1.TSK
File Transfer (F) or System Image Transfer (S)? S
In which unit is your cartridge mounted? 0
Starting Block Number on Tape (0-511)? 72

5  BLOCKS WRITTEN
Number of Blocks Written = 7
Approx. Size of S.M.T. Image = 3K Bytes
```

USER TASK 2.

```
Source File Name? MAZ3.TSK
File Transfer (F) or System Image Transfer (S)? S
In which unit is your cartridge mounted? 0
Starting Block Number on Tape (0-511)? 80
Number of Blocks Written = 2
Approx. Size of S.M.T. Image = 1K Bytes
Source File Name? MAZ4.TSK
File Transfer (F) or System Image Transfer (S)? S
In which unit is your cartridge mounted? 0
Starting Block Number on Tape (0-511)? 83
Number of Blocks Written = 4
Approx. Size of S.M.T. Image = 2K Bytes
Source File Name? LMAZ5.TSK
File Transfer (F) or System Image Transfer (S)? S
In which unit is your cartridge mounted? 0
Starting Block Number on Tape (0-511)? 88
5  BLOCKS WRITTEN
10  BLOCKS WRITTEN
15  BLOCKS WRITTEN
```

ETC

Figure E8

LOADING OF AN EDITED TASK ON LINE