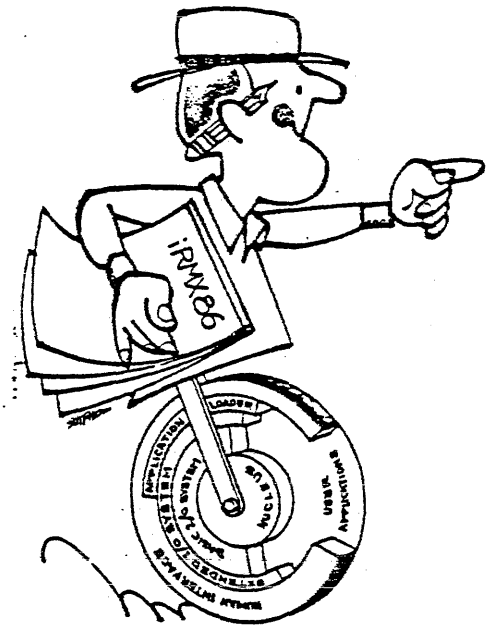
iRMX 86



VERSION 4.0

INTEL CORPORATION

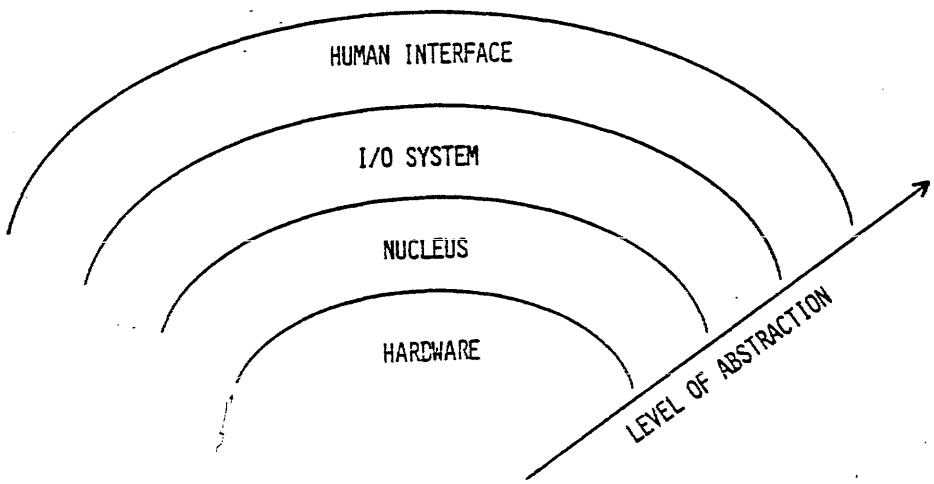
MARCH 1982

iRMX 86 OVERVIEW

WHAT iRMX 86 PROVIDES

- ABSTRACTION OF MACHINE FUNCTIONS
- SEPARATION OF PROGRAMMER SKILLS, CONCENTRATION ON MODULAR DESIGN
- RANGE OF PRE WRITTEN SOFTWARE
  - I/O SYSTEMS
  - TASK CONTROL
  - EXECUTIVE CONTROL SOFTWARE
- AN ENVIRONMENT FOR RUNNING MANY PROGRAMS
  - EFFICIENT USE OF RESOURCES
  - PREVENTION OF DEADLOCKS
  - SCHEDULING
  - CHANGE OF CONTEXT

LAYERING OF AN O.S.

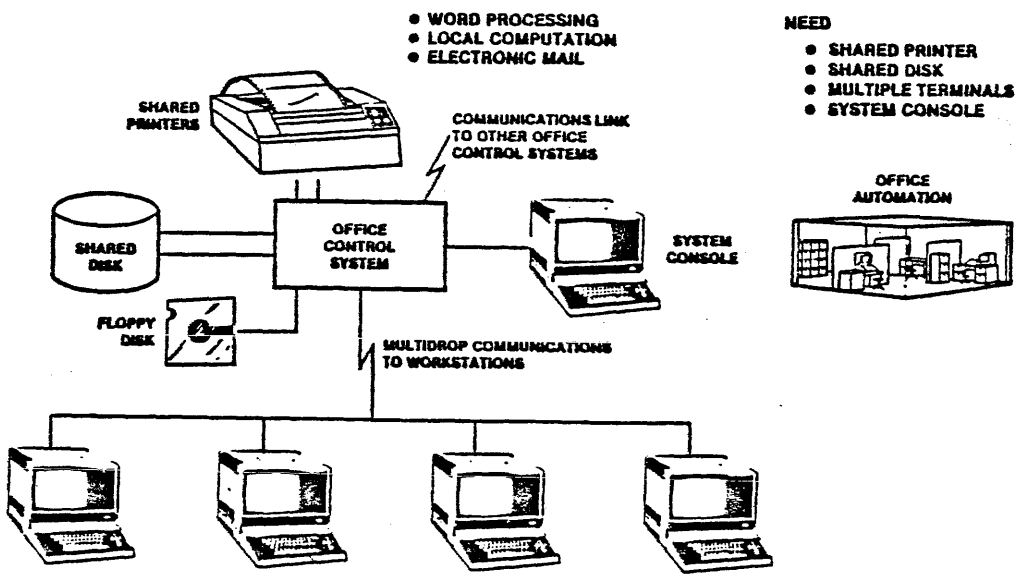


Q. WHEN IS A REAL-TIME MULTI-TASKING EXECUTIVE REQUIRED?

A. WHEN THE APPLICATION NEEDS:

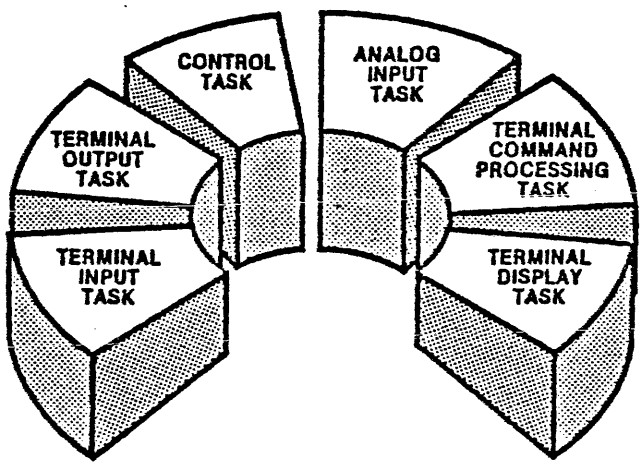
- MULTIPLE CONCURRENT PROCESSES
- MULTIPLE ASYNCHRONOUS EVENTS

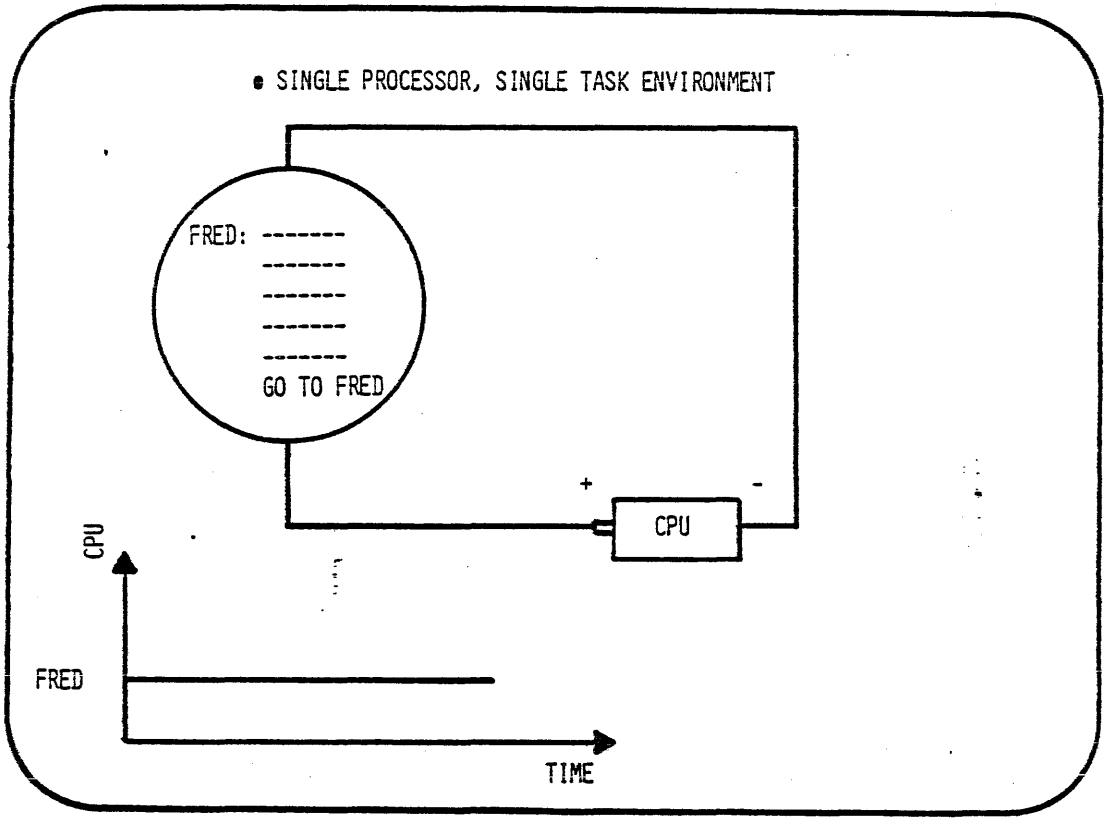
EXAMPLE: OFFICE AUTOMATION



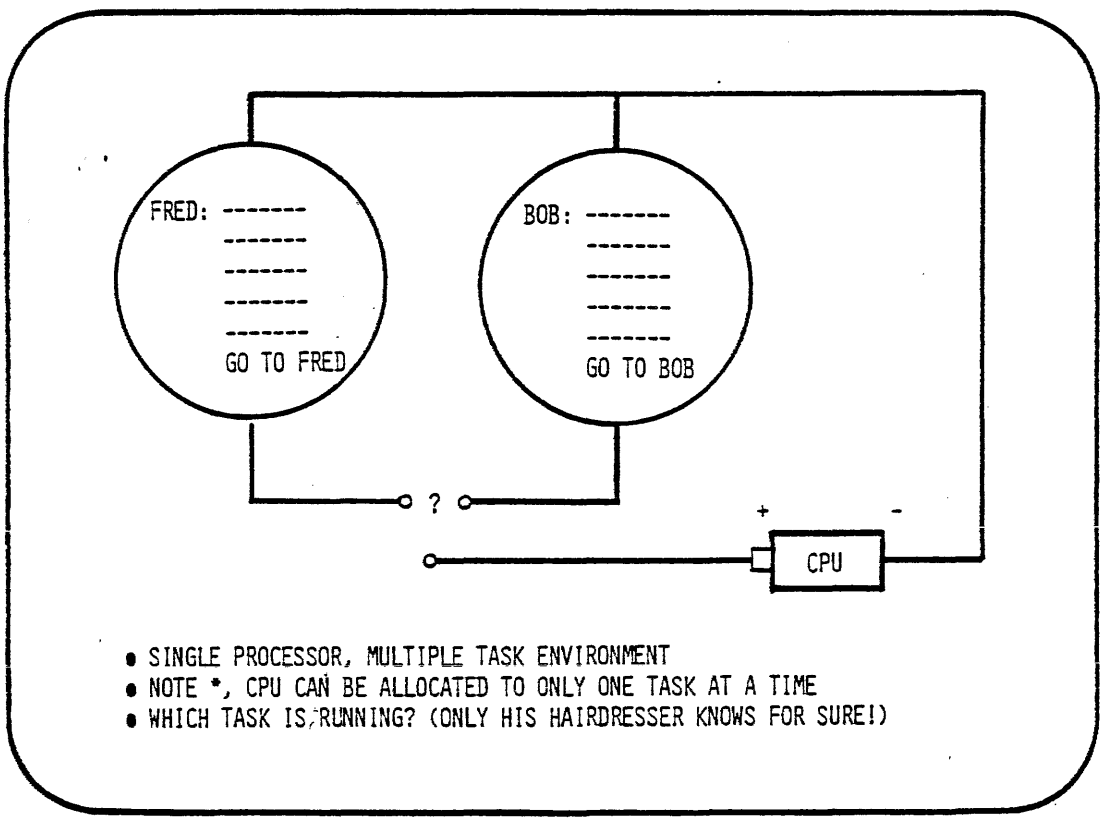
SIMPLE TASKS

THE APPLICATION SOFTWARE IS
DIVIDED INTO A NUMBER OF TASKS.




• SINGLE PROCESSOR, SINGLE TASK ENVIRONMENT
FRED: -------

GO TO FRED
+
-
CPU
CPU
FRED
TIME


FRED: -------

GO TO FRED
BOB: -------

GO TO BOB
?
+
-
CPU
• SINGLE PROCESSOR, MULTIPLE TASK ENVIRONMENT
• NOTE *, CPU CAN BE ALLOCATED TO ONLY ONE TASK AT A TIME
• WHICH TASK IS RUNNING? (ONLY HIS HAIRDRESSER KNOWS FOR SURE!)

CPU IS FREE

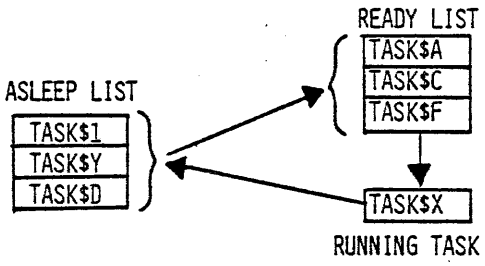
- AN RQSLEEP SYSTEM CALL IS NOT A SOFTWARE DELAY LOOP
- IF BOTH TASKS ARE ASLEEP AT THE SAME TIME THE O.S. PROVIDES A DEFAULT TASK THAT DOES NOTHING
- THE CPU, UNDER THE CONTROL OF THE O.S., IS ALWAYS RUNNING A TASK

EXERCISE

TASK A: WRITE A PLM TASK TO ACCOMODATE THE FOLLOWING FLOWCHART

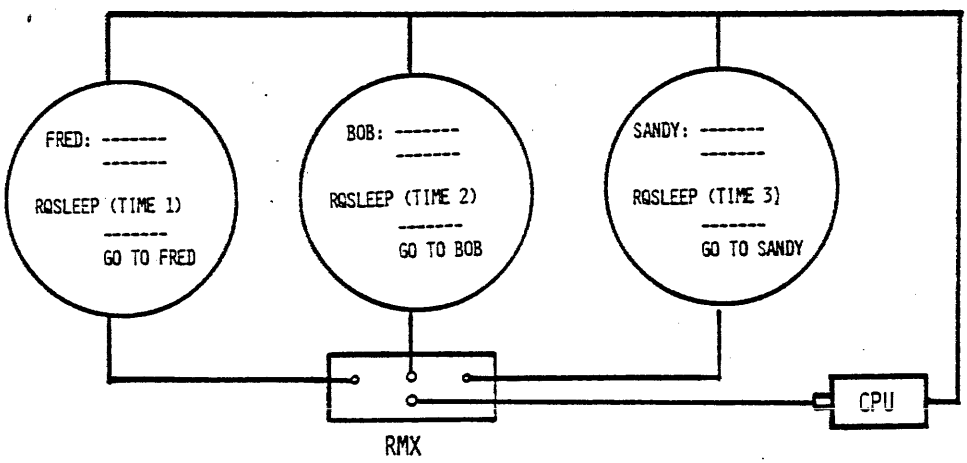
1. INITIALIZE A BYTE VARIABLE TO 1
2. OUTPUT VARIABLE TO PORT Ø
3. GO TO SLEEP FOR 1/4 SEC
4. ROTATE VARIABLE TO THE LEFT BY ONE
5. GO TO 2.

- TASK STATES



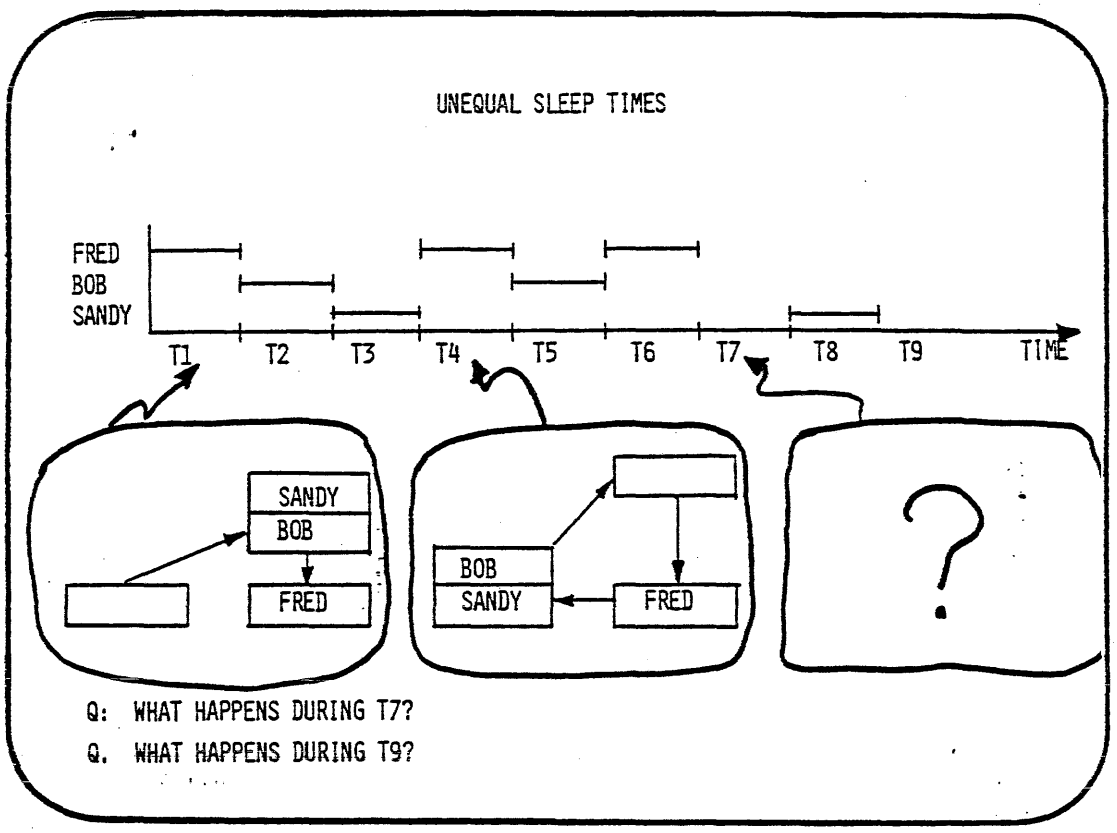
- THERE MAY BE MORE THAN ONE TASK IN THE READY AND ASLEEP STATES

- EXAMPLE OF 3 TASKS



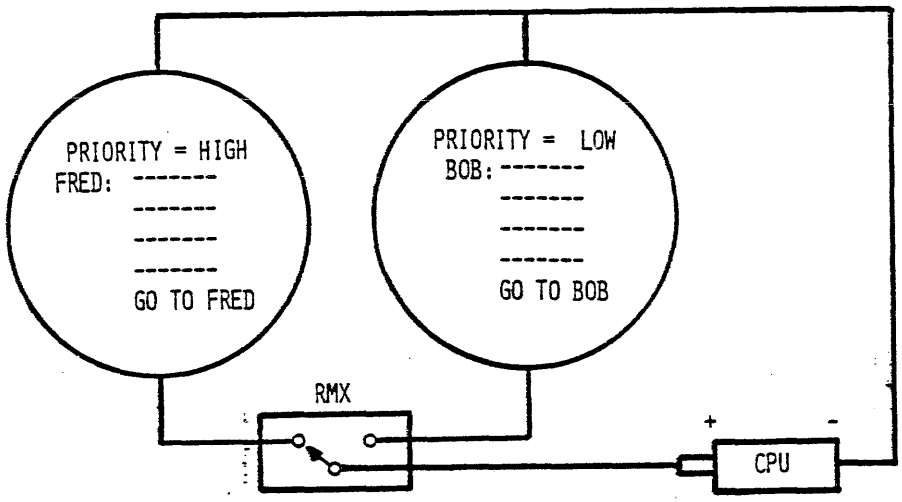
CASE 1:

- GIVEN:
  - TIME 1 = TIME 2 = TIME 3 EQUAL EXECUTION TIMES ON ALL TASKS
  - INITIAL RUNNING TASK = FRED
  - INITIAL READY LIST = BOB, SANDY



Q: HOW CAN WE GAIN MORE CONTROL OVER THE BEHAVIOR OF THE TASKS IN OUR APPLICATION?

UNEQUAL PRIORITIES



- BY ASSIGNING PRIORITIES FRED RUNS.
- WILL BOB EVER RUN?

THE RUNNING TASK

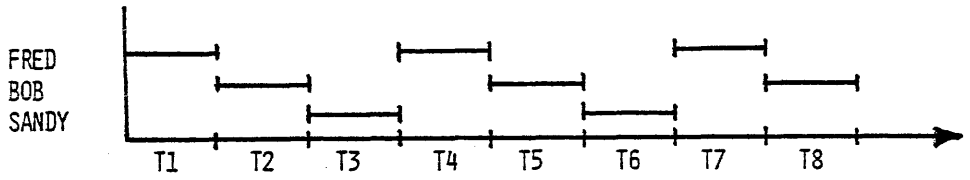
- NOTE* FOR A TASK TO BE THE RUNNING TASK TWO CONDITIONS MUST BE MET.

  1) A TASK MUST BE READY TO RUN

  2) IT MUST BE THE HIGHEST PRIORITY TASK

- PRIORITY ALONE DOES NOT PROVIDE THE SOLUTION

PRIORITIZED TASKS, EQUAL SLEEP TIMES



- LET'S TAKE CASE 1 AGAIN:

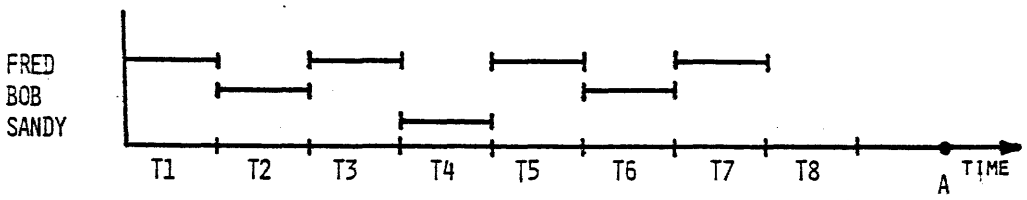
  GIVEN: TIME1 = TIME2 = TIME3 EQUAL EXECUTION AND SLEEP TIMES ON ALL TASKS

  Q: WHAT WILL THE INITIAL READY LIST BE?

- FRED AND BOB NEVER RELINQUISH CONTROL
- SANDY NEVER RUNS

- LET'S TAKE CASE 2 AGAIN:

  GIVEN: - SLEEP TIMES
  - TIME1 = 1, TIME2 = 2, TIME3 = 4
  - EXECUTION TIMES ALL TASKS = 1



- FRED WILL ALWAYS RUN WHEN HE SAKES UP. WHY?

Q: WHAT TASK RUNS DURING T8?

Q: WHAT HAPPENS AT POINT A?

ADVANCED TASK TOPICS

- A TASK IS:

  -SCHEDUABLE UNIT OF WORK

- AT ANY POINT IN TIME A TASK CAN BE DEFINED BY
  - CURRENT PROGRAM COUNTER (CS, IP 8086 REGISTERS)
  - CURRENT STACK POINTER (SS, SP 8086 REGISTERS)
  - TASK PRIORITY (0-255)
  - TASK STATE (RUNNING, READY, ASLEEP . . .)
  - REGISTERS (OTHER 8086 REGISTERS AND/OR NDP REGISTERS)

- PARAMETERLESS, UNTYPED, PUBLIC PROCEDURE THAT NEVER TERMINATES, (UNLESS THE TASK GETS DELETED)

- THE MODULE IN PLM 86 MUST BE A NON-MAIN MODULE

TASK CREATION

- CREATION OF A TASK IS ACCOMPLISHED AT RUN TIME BY AN RQ$CREATE$TASK SYSTEM CALL

- THE TASK'S CODE MUST RESIDE IN SYSTEM MEMORY AT THE TIME THE CALL IS MADE

- AFTER THE CALL, THE O.S. RETURNS AN IDENTIFIER NUMBER TO THE CREATOR

- THIS NUMBER IS CALLED A TOKEN

PREEMPTION

- AS A TASK FROM THE READY LIST BECOMES THE RUNNING TASK, THE FOLLOWING EVENTS OCCUR, IN ORDER:
  - THE VALUES (CPU REGISTERS, ETC.) OF THE PREVIOUSLY RUNNING TASK ARE SAVED BY THE O.S.
  - THE O.S. LOADS THE NEW RUNNING TASK'S VALUES

  } CONTEXT SWITCH

  - THE NEW TASK BEGINS EXECUTING

- EXAMPLE:
  - IF THE RUNNING TASK CREATES A HIGHER PRIORITY TASK, RELATIVE TO ITSELF, THEN THE RUNNING TASK IS PREEMPTED BY THE NEWLY CREATED TASK.

TASK DELETION

IF A TASK IS NO LONGER NEEDED, THEN IT CAN BE DELETED

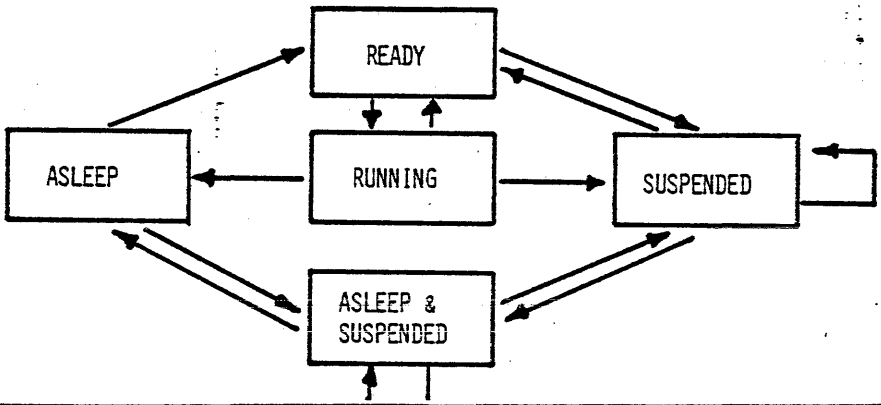
- THE FORM OF THE CALL IS

```
CALL RQ$DELETE$TASK(TASK,EXCEPT$PTR);
```

- REFER TO NUCLEUS REFERENCE MANUAL FOR A DETAILED DESCRIPTION OF PARAMETERS

SUSPENDED STATE

- THE RUNNING TASK MAY SUSPEND ITSELF OR ANOTHER TASK
- THE RUNNING TASK MAY RESUME ANOTHER TASK
- A TASK MAY BE ASLEEP AND SUSPENDED
- A TASK MAY BE SUSPENDED MORE THAN ONCE



TASK SUSPENSION

- THE FORM OF THE CALL IS:

```
CALL RQ$SUSPEND$TASK (TASK, EXCEPT$PTR);
```

- THE O.S. INCREMENTS THE SUSPENSION DEPTH OF THE TASK BY ONE EACH TIME THE CALL IS MADE.

INTERRUPTS

QUIZ

- MATCH THE DESCRIPTION TO THE SYSTEM CALL

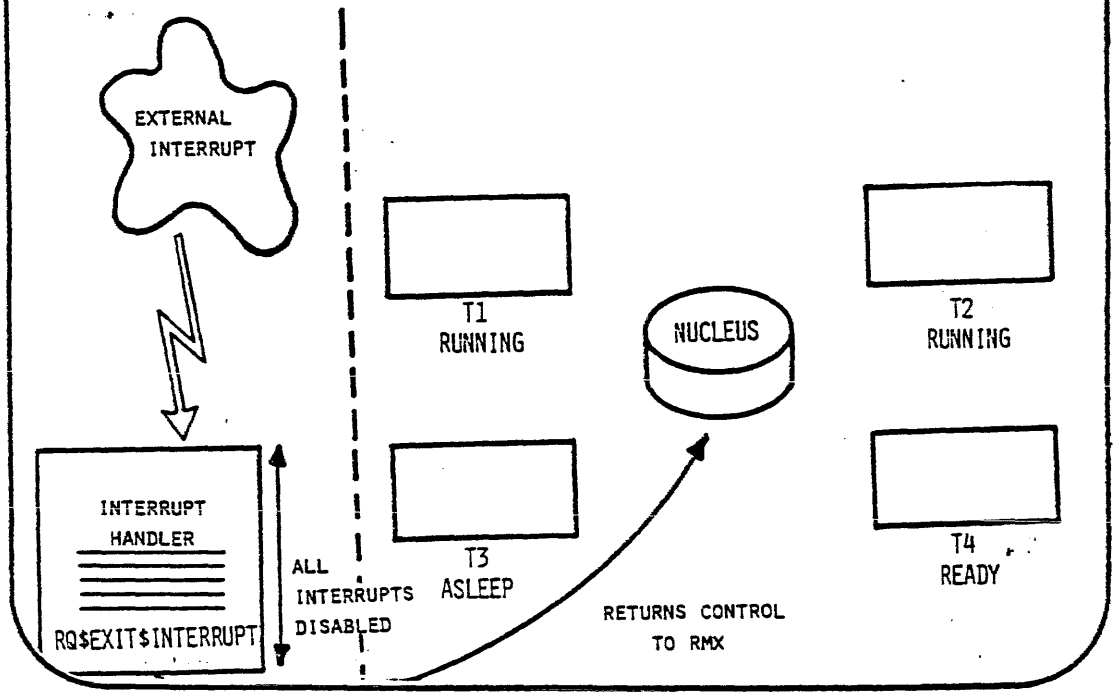
RQ$DELETE$TASK	___	(A)-INCREASES A TASK'S SUSPENSION DEPTH BY ONE; SUSPENDS THE TASK IF IT IS NOT ALREADY SUSPENDED
RQ$CREATE$TASK	___	(B)-CREATES A TASK AND RETURNS A TOKEN FOR IT
RQ$SUSPEND$TASK	___	(C)- DELETES A TASK FROM THE SYSTEM
RQ$RESUME$TASK	___	(D)- PLACES THE CALLING TASK IN THE ASLEEP STATE FOR A SPECIFIED AMOUNT OF TIME
RQ$SLEEP	___	(E) -DECREASES A TASK'S SUSPENSION DEPTH BY ONE; IF THE DEPTH BECOMES ZERO AND THE TASK WAS SUSPENDED, IT THEN BECOMES READY; IF THE DEPTH BECOMES ZERO AND THE TASK WAS ASLEEP-SUSPENDED, THEN IT GOES INTO THE ASLEEP STATE

INTERRUPT SERVICE

- AN INTERRUPT IS SERVICED IN ONE OF TWO WAYS
  - AN INTERRUPT HANDLER SERVICES THE INTERRUPT ALONE
  - AN INTERRUPT HANDLER INVOKES AN INTERRUPT TASK
- IN PLM 86 INTERRUPT HANDLERS ARE WRITTEN AS INTERRUPT PROCEDURES

INTERRUPT HANDLER ALONE

- THE INTERRUPT HANDLER SERVICES THE INTERRUPT OUTSIDE OF THE CONTROL OF RMX



IDENTIFYING INTERRUPT HANDLERS TO RMX

- INTERRUPT HANDLERS ARE SPECIFIED TO RMX BY THE SET$INTERRUPT SYSTEM CALL

```
INITIALIZE$TASK: PROCEDURE;
   CALL RQ$SET$INTERRUPT(LEVEL, INTERRUPT$TASK$FLAG,
                         INTERRUPT$PTR(INT$HANDLER$),...);
   END INITIALIZE$TASK;
```

  - THE FIRST PARAMETER INDICATES THE 8086 INTERRUPT LEVEL
  - THE INTERRUPT$TASK$FLAG PARAMETER INDICATES IF THERE IS TO BE AN INTERRUPT TASK ASSOCIATED WITH THIS INTERRUPT LEVEL

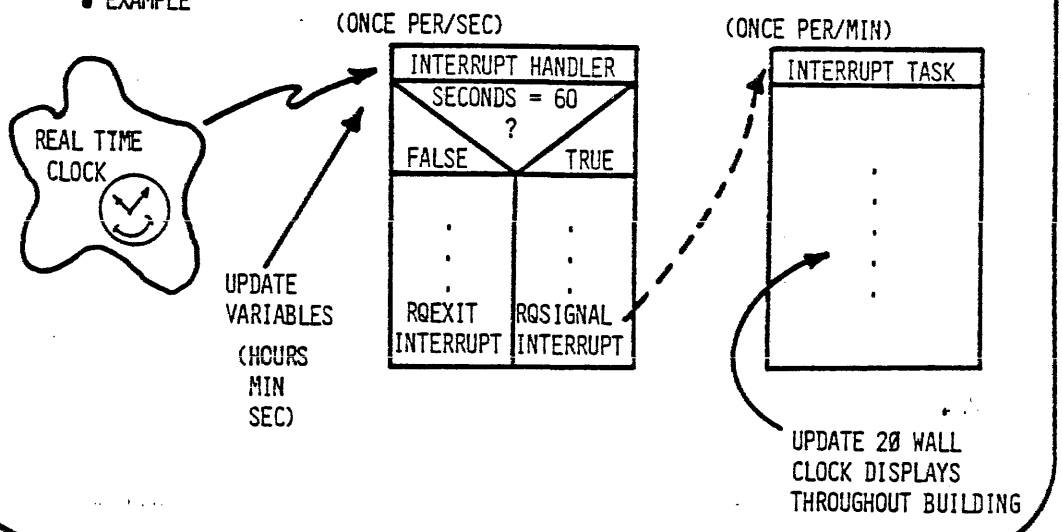
    INTERRUPT$TASK$FLAG = 0 THEN NO INTERRUPT TASK

  - USE PLM86 INTERRUPT$PTR BUILT-IN TO PASS THE STARTING ADDRESS OF THE INTERRUPT HANDLER TO RMX

- WHEN THE INTERRUPT HANDLER IS IDENTIFIED TO RMX
  - RMX ENTERS THE HANDLER'S ADDRESS INTO THE INTERRUPT VECTOR TABLE. (THE PROCEDURE SHOULD BE COMPILED WITH NO$INTVECTOR)
  - RMX ENABLES THE CORRESPONDING LEVEL OF INTERRUPTS IN THE HARDWARE

USING INTERRUPT TASKS

- AN INTERRUPT TASK WILL BE NEEDED WHEN
  - ANY SYSTEM CALLS ARE NEEDED TO SERVICE THE INTERRUPT
  - SYSTEM INTERRUPT LATENCY TIME IS SHORTER THAN THE EXECUTION TIME OF THE INTERRUPT HANDLER. (HOW LONG CAN YOU AFFORD TO WAIT?)
- EXAMPLE



- A TASK UNMASKS ITS LEVEL OF INTERRUPT BY A WAIT$INTERRUPT SYSTEM CALL

  - THIS CAUSES THE TASK TO WAIT FOR THE INTERRUPT HANDLER TO EXECUTE AN RQSIGNAL SYSTEM CALL.

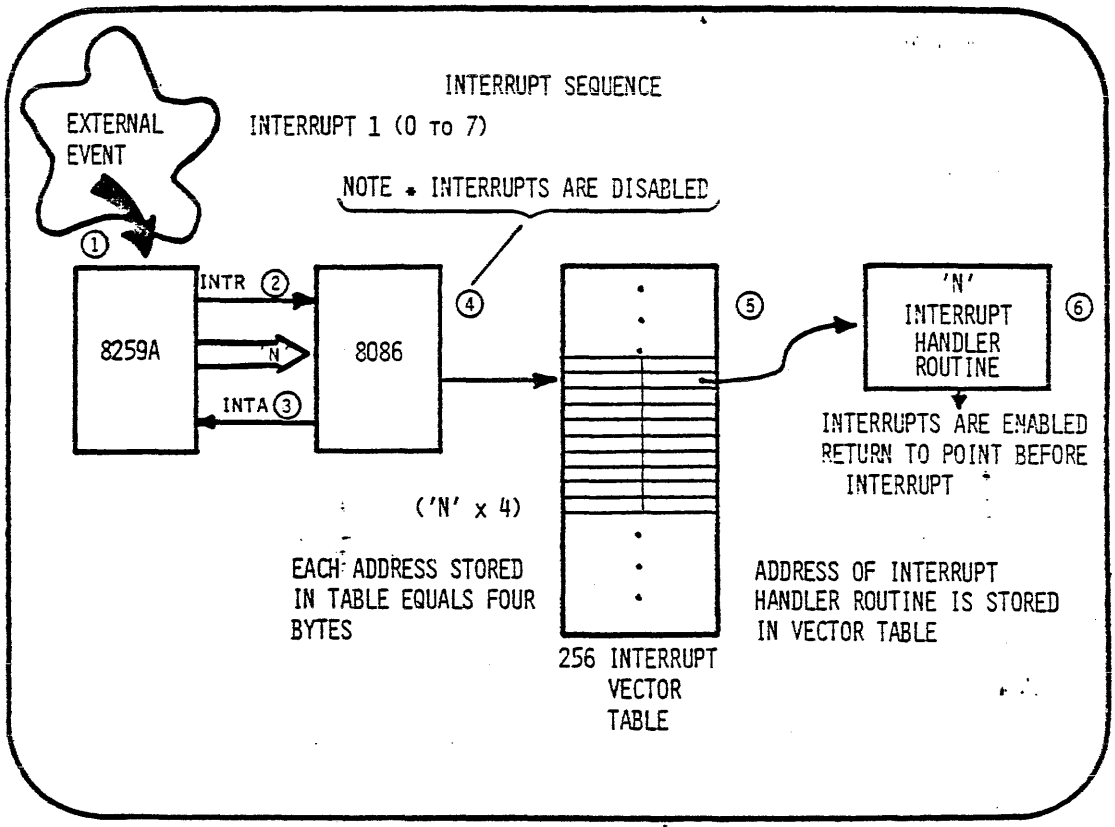
```
INT$TASK: PROCEDURE;
     CALL RQ$SET$INTERRUPT(...);
        :
     DO FOREVER;
          CALL RQ$WAIT$INTERRUPT(...);
             :
```

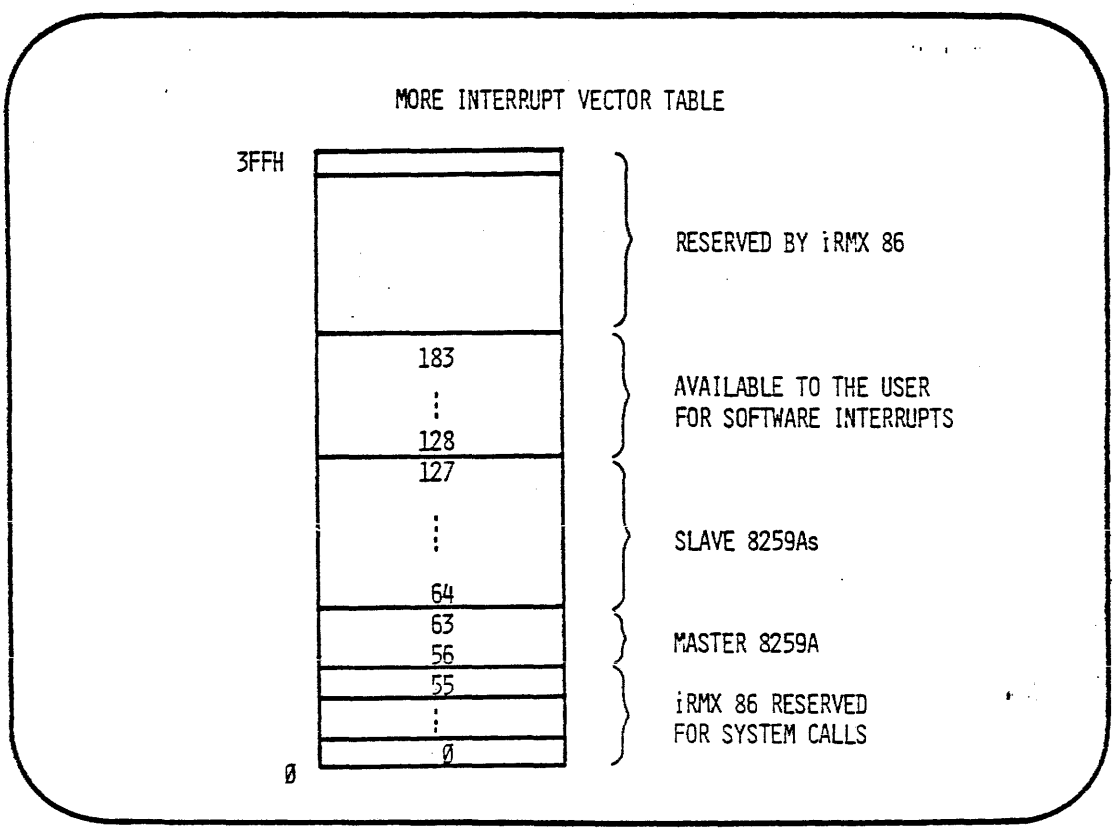
- AFTER INITIALIZATION, AN INTERRUPT TASK IS ALWAYS EITHER SERVICING AN INTERRUPT OR WAITING FOR AN INTERRUPT

QUIZ

- NAME TWO DIFFERENT SOFTWARE STRATEGIES FOR SERVING INTERRUPTS.

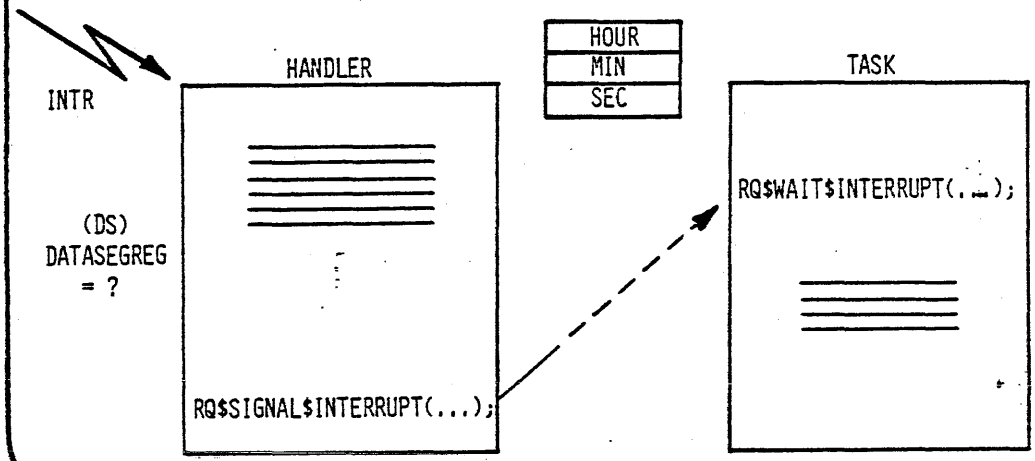
- FOR FAST INTERRUPT RESPONSE, THE USER CAN USE THE INTERRUPT HANDLER ALONE. NAME ONE DISADVANTAGE AND ONE ADVANTAGE.


INTERRUPT SEQUENCE
EXTERNAL EVENT
INTERRUPT 1 (0 TO 7)
NOTE * INTERRUPTS ARE DISABLED
1
8259A
INTR 2
N
INTA 3
8086
4
5
'N' INTERRUPT HANDLER ROUTINE
6
INTERRUPTS ARE ENABLED RETURN TO POINT BEFORE INTERRUPT
('N' x 4)
EACH ADDRESS STORED IN TABLE EQUALS FOUR BYTES
ADDRESS OF INTERRUPT HANDLER ROUTINE IS STORED IN VECTOR TABLE
256 INTERRUPT VECTOR TABLE


MORE INTERRUPT VECTOR TABLE
3FFH
RESERVED BY iRMX 86
183
128
AVAILABLE TO THE USER FOR SOFTWARE INTERRUPTS
127
64
SLAVE 8259As
63
56
MASTER 8259A
55
Ø
iRMX 86 RESERVED FOR SYSTEM CALLS
Ø

PROBLEM!

- HOW CAN THE INTERRUPT HANDLER SHARE THE HOURS, MIN, SEC VARIABLES WITH THE INTERRUPT TASK?



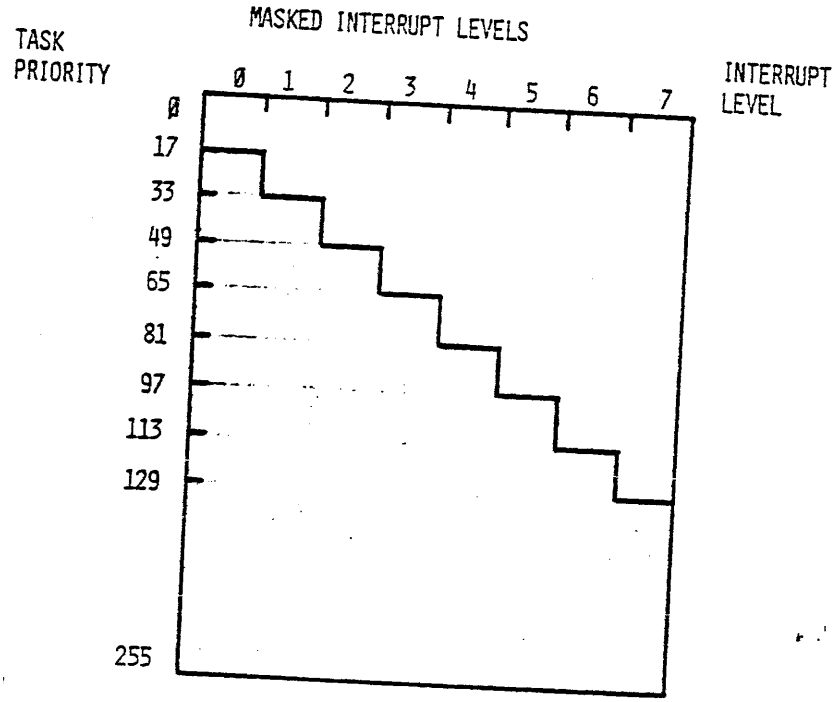
FIRST STEP: WHEN THE INTERRUPT TASK IDENTIFIES ITSELF TO THE RMX O.S. THE DATA SEGMENT IS PASSED AS A PARAMETER.

```
CALL RQ$SET$INTERRUPT(,,,DATA$SEG,....);
```

TASK PRIORITY GROUPS

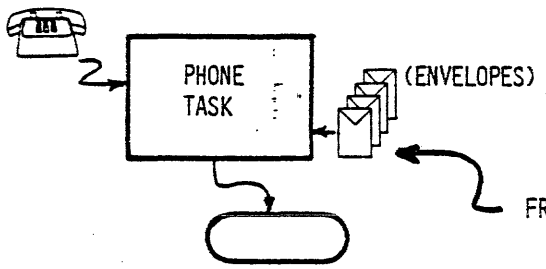
- TASKS ARE ASSIGNED A PRIORITY NUMBER FROM Ø TO 255
- TASK PRIORITIES Ø TO 128 ARE SPLIT INTO GROUPS OF 16
- EACH GROUP IS ASSOCIATED WITH AN 8259A INTERRUPT LEVEL

PRIORITY GROUPS	8259A LEVEL
Ø - 16	Ø
17 - 32	1
33 - 48	2
49 - 64	3
65 - 80	4
81 - 96	5
97 -112	6
113 - 128	7
129 - 255	NONE



PHONE TASK

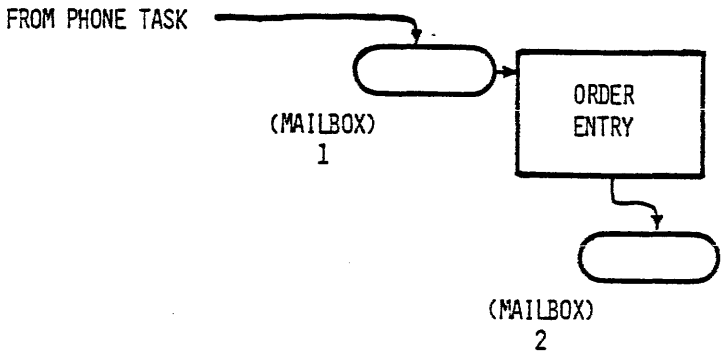
1 - WAIT FOR PHONE TO RING
2 - GET ORDER(S) FROM CUSTOMER
3 - GET EMPTY ENVELOPE(S) FROM ENVELOPE RACK
4 - PLACE ORDER(S) INSIDE ENVELOPE(S)
5 - SEND ENVELOPE(S) TO MAILBOX 1
6 - GO TO 1



COMPANY RULES:
- ONLY ONE ORDER PER ENVELOPE
- MAY HAVE MANY ORDERS PER CUSTOMER

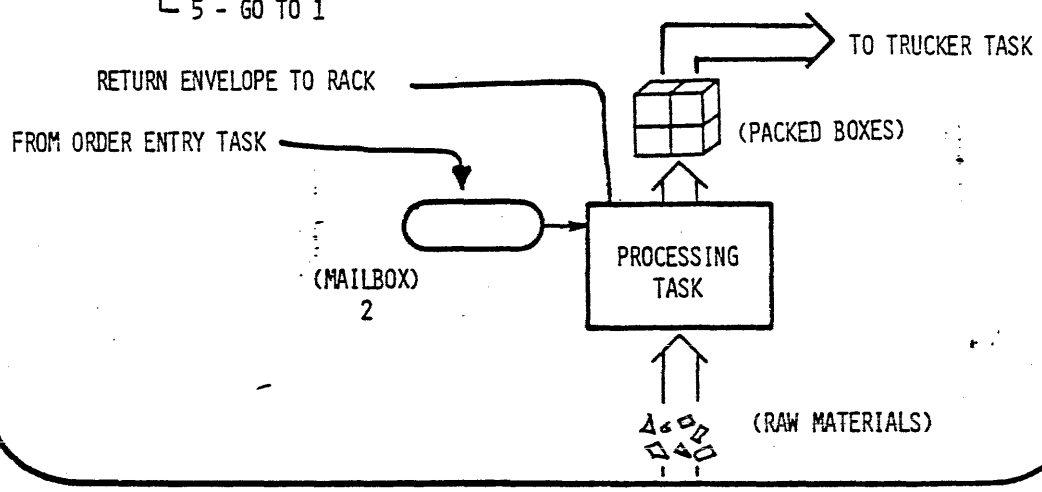
ORDER ENTRY TASK

1 - WAIT FOR ENVELOPE IN MAILBOX 1
2 - LOG THE ORDER IN COMPANY BOOKS
3 - SEND ENVELOPE TO MAILBOX 2.
4 - GO TO 1



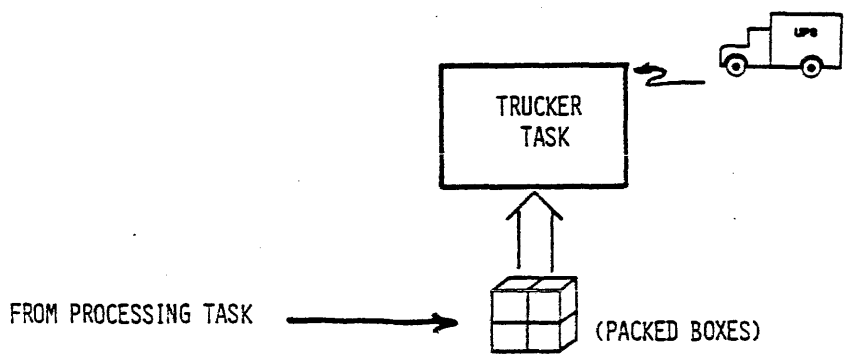
PROCESSING TASK

1 - WAIT FOR AN ENVELOPE IN MAILBOX 2
2 - GET RAW MATERIALS TO FILL ORDER, ASSEMBLE AND PACK WIDGETS
3 - RETURN ENVELOPE TO ENVELOPE RACK
4 - CALL PARCEL SERVICE COMPANY FOR PICKUP
5 - GO TO 1



TRUCKER TASK

1 - WAIT FOR TRUCK TO ARRIVE
2 - OPEN SHIPPING DOCK
3 - GIVE BOXES AND INSTRUCTIONS TO TRUCKER
4 - CLOSE SHIPPING DOCK
5 - GO TO 1



INTERCOMMUNICATION RULES

- INTERCOMMUNICATION
  - ENVELOPES CARRY ORDERS FROM TASK, TO MAILBOX, TO TASK
  - NOTE ENVELOPES DO NOT GO DIRECTLY FROM TASK TO TASK

- ENVELOPES
  - THERE IS A FINITE NUMBER OF ENVELOPES
  - THEY MUST BE RETURNED WHEN NOT IN USE

INTER-TASK COMMUNICATION

OFFICE WORK FLOW

ASSUME MANY ENVELOPES ARE WAITING TO BE PROCESSED. TWO ALTERNATIVES:

(A) 1. AN ENVELOPE IS PROCESSED AT A DESK AND SENT TO THE NEXT MAILBOX.

2. THE MAN THEN MOVES IMMEDIATELY TO THE NEXT DESK.

(B) 1. AN ENVELOPE IS PROCESSED AT A DESK AND SENT TO THE NEXT MAILBOX.

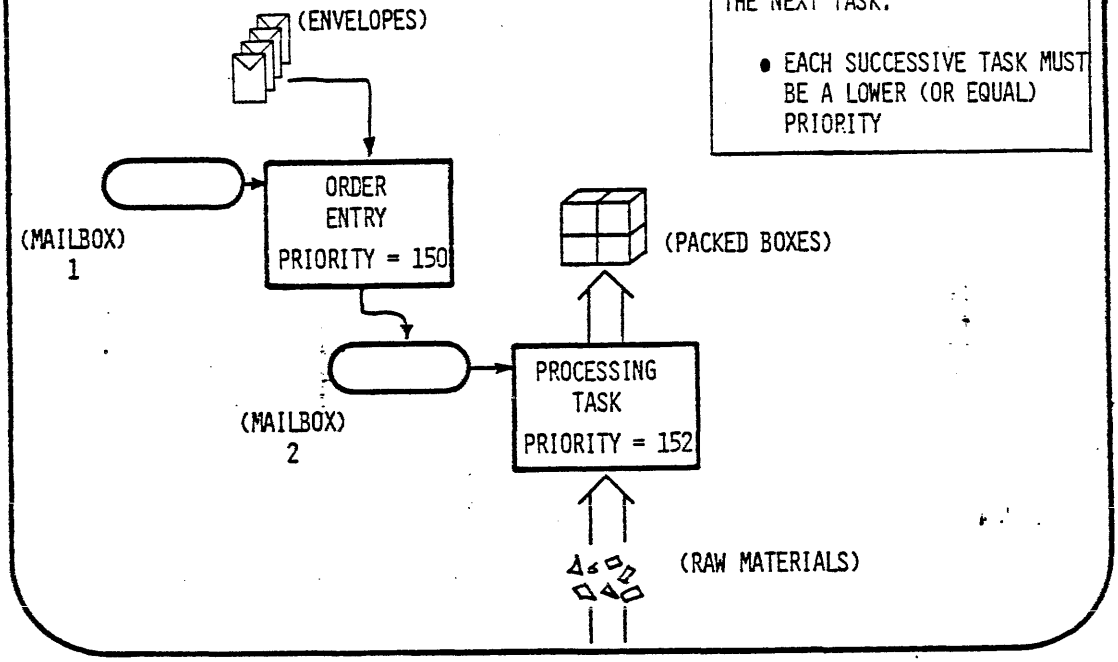
2. THE NEXT ENVELOPE IS PROCESSED, AND MOVED. THIS CONTINUES UNTIL NO MORE ENVELOPES NEED TO BE PROCESSED.

3. THE MAN MOVES TO THE NEXT DESK.

ALTERNATIVE B

TO COMPLETE ONE TASK FOR ALL ENVELOPES BEFORE STARTING THE NEXT TASK.

- EACH SUCCESSIVE TASK MUST BE A LOWER (OR EQUAL) PRIORITY

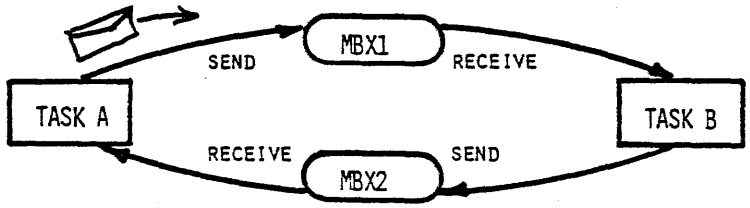


SYSTEM IS EVENT DRIVEN

EVENT		TASK
PHONE RINGS	INTERRUPT →	PHONE TASK
MAILBOX 1 CONTAINS ENVELOPE	SEND/RECEIVE →	ORDER ENTRY TASK
MAILBOX 2 CONTAINS ENVELOPE	SEND/RECEIVE →	PROCESSING TASK
TRUCK ARRIVES	INTERRUPT →	TRUCKER TASK
ALARM GOES ON	INTERRUPT →	WATER PLANT TASK
NOTHING ELSE TO DO	DEFAULT →	DRINK COFFEE TASK

SEND AND RECEIVE EXAMPLE

- TASK A WILL SEND A MESSAGE TO TASK B THROUGH A MAILBOX CALLED MBX1.
- THEN TASK A WILL WAIT (RECEIVE) AT A SECOND MAILBOX CALLED MBX2 BEFORE IT CONTINUES EXECUTING.



- TASK B WILL WAIT (RECEIVE) AT MBX1 FOR MESSAGE.
- WHEN TASK B RECEIVES THE MESSAGE IT WILL PROCESS THE INFORMATION IN THE MESSAGE.
- THEN TASK B WILL SEND THE SAME MESSAGE (FOR SYNCHRONIZATION) TO MBX2 WHERE TASK A IS WAITING.

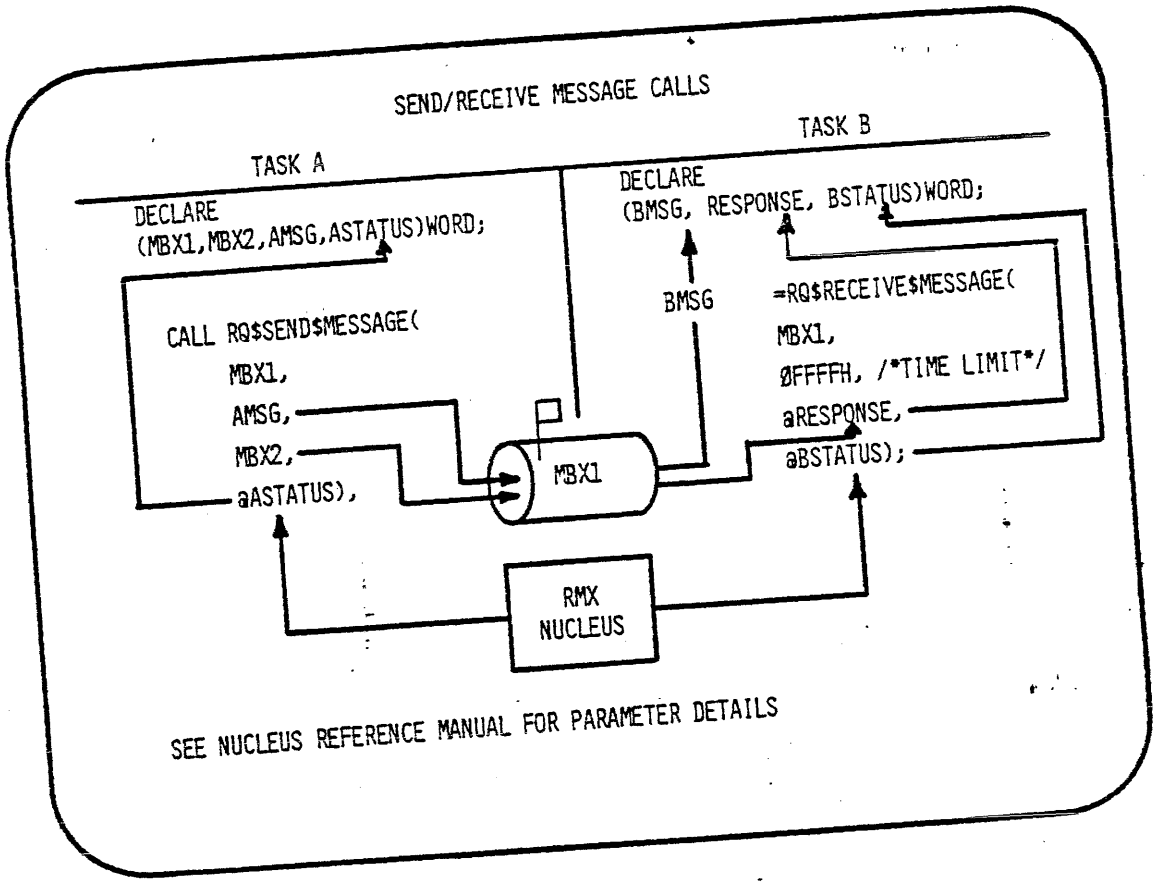
SEND A MESSAGE

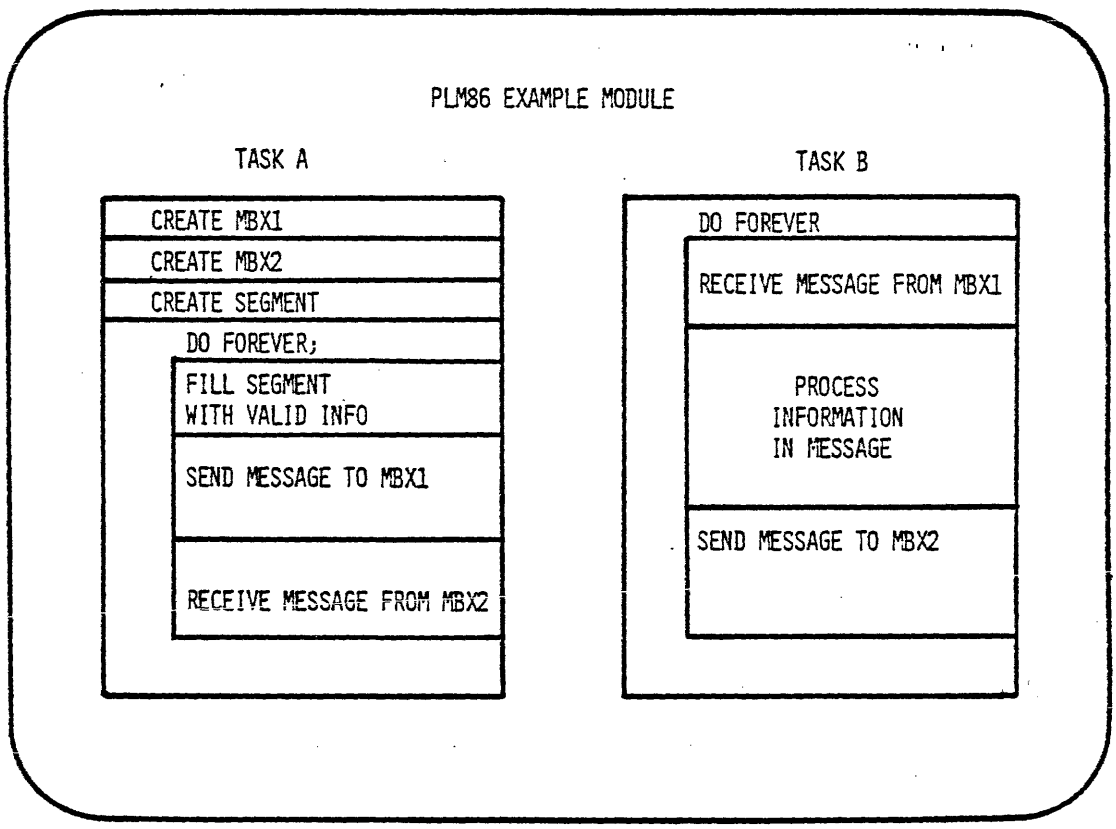
- THE FORM OF THE CALL IS

```
CALL RQ$SEND$MESSAGE (WHERE, WHAT, ...);
```

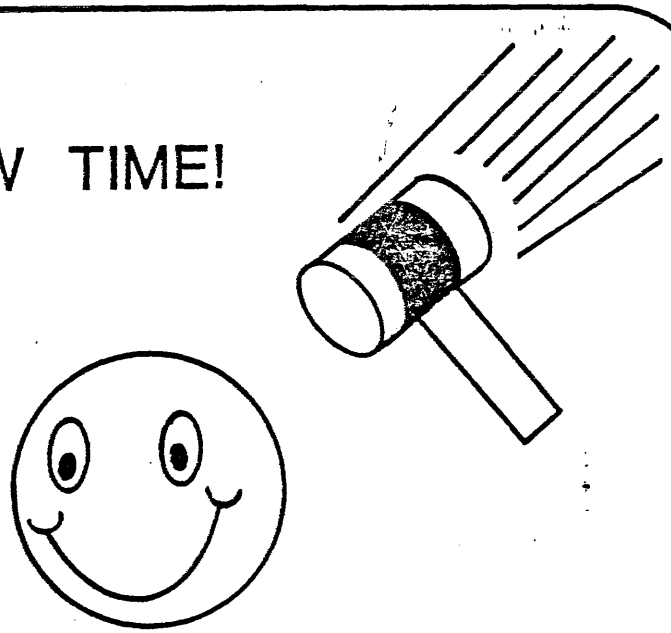
WHERE? = MBX1

WHAT? = ENVELOPE


SEND/RECEIVE MESSAGE CALLS
TASK A
TASK B
DECLARE
(MBX1,MBX2,AMSG,ASTATUS)WORD;
DECLARE
(BMSG, RESPONSE, BSTATUS)WORD;
CALL RQ$SEND$MESSAGE(
MBX1,
AMSG,
MBX2,
@ASTATUS),
BMSG
=RQ$RECEIVE$MESSAGE(
MBX1,
0FFFFH, /*TIME LIMIT*/
@RESPONSE,
@BSTATUS);
MBX1
RMX
NUCLEUS
SEE NUCLEUS REFERENCE MANUAL FOR PARAMETER DETAILS


PLM86 EXAMPLE MODULE
TASK A
CREATE MBX1
CREATE MBX2
CREATE SEGMENT
DO FOREVER;
FILL SEGMENT
WITH VALID INFO
SEND MESSAGE TO MBX1
RECEIVE MESSAGE FROM MBX2
TASK B
DO FOREVER
RECEIVE MESSAGE FROM MBX1
PROCESS
INFORMATION
IN MESSAGE
SEND MESSAGE TO MBX2

REVIEW TIME!



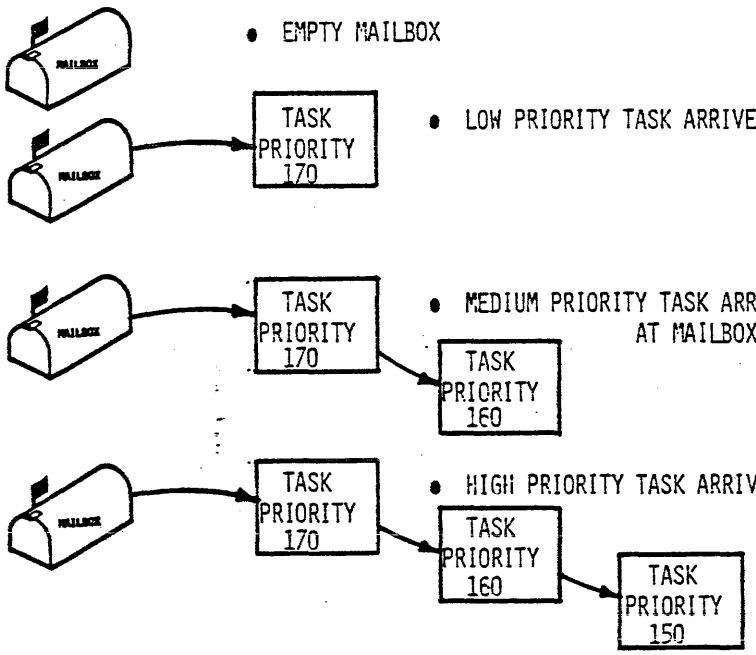
QUIZ

- MATCH THE DESCRIPTION TO THE SYSTEM CALL.

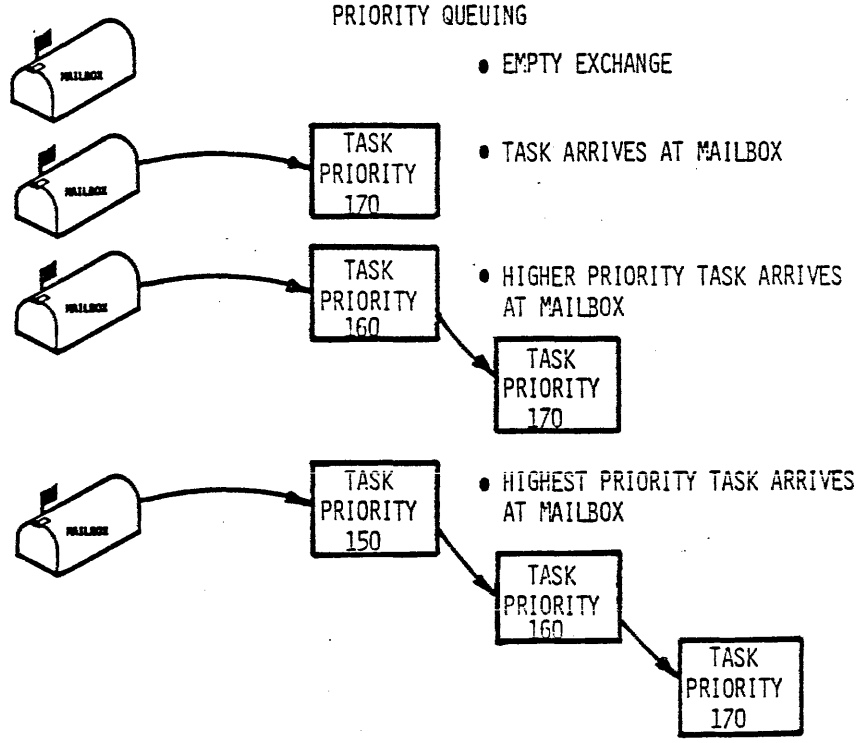
RQ$CREATE$SEGMENT ____	Ⓐ	THE CALLING TASK WAITS AT A MAILBOX
RQ$DELETE$MAILBOX ____	Ⓑ	SEND AN OBJECT TO A MAILBOX
RQ$RECEIVE$MESSAGE____	Ⓒ	CREATES A SEGMENT AND RETURNS A TOKEN FOR IT
RQ$DELETE$SEGMENT ____	Ⓓ	CREATES A MAILBOX AND RETURNS A TOKEN FOR IT
RQ$SEND$MESSAGE ____	Ⓔ	RETURNS A SEGMENT TO THE POOL FROM WHICH IT WAS ALLOCATED
RQ$CREATE$MAILBOX ____	Ⓕ	RETURNS THE SIZE, IN BYTES, OF A SEGMENT
RQGETSIZE ____	Ⓖ	DELETES A MAILBOX FROM THE SYSTEM

FIFO QUEUING

REGARDLESS OF PRIORITY, FIRST TASK TO ARRIVE RECEIVES FIRST OBJECT TO ARRIVE.

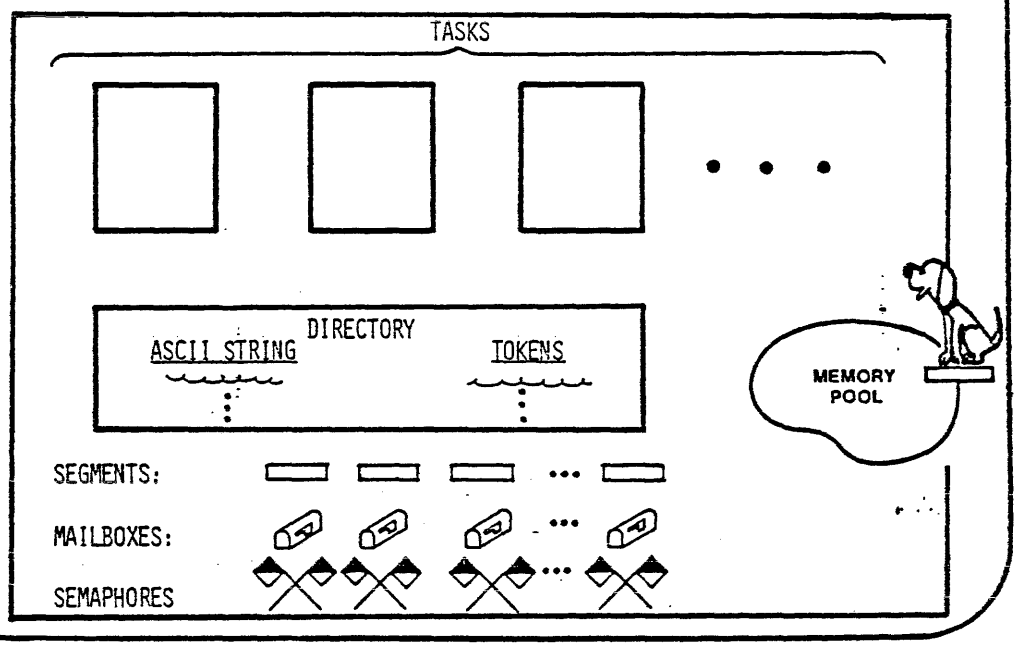


PRIORITY QUEUING



JOB

- A JOB CONSISTS OF TASKS AND THE OBJECTS THEY CREATE FROM THE JOB'S MEMORY POOL



SHARING RESOURCES

- IN AN EARLIER EXAMPLE THE TOKEN FOR MAILBOX 1 WAS GLOBAL AND THEREFORE COULD BE USED BY BOTH.

```
DECLARE (MBX1)...
TASK$A
   DECLARE ( , , )...
TASK$B
   DECLARE ( , , )...
```

JOB DIRECTORY

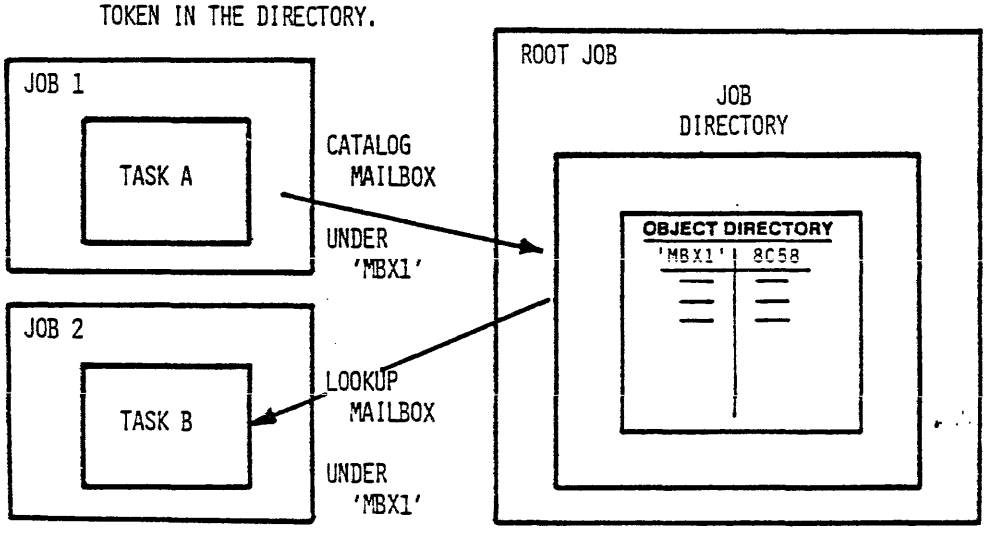
- A DIRECTORY ALLOWS JOBS TO SHARE RESOURCES (TASKS, SEGMENTS, MAILBOXES...)
- A RESOURCE IS ENTERED INTO THE DIRECTORY (CATALOGED) BY CALLING RQ$CATALOG$OBJECT
- OBJECTS ARE REFERENCED BY ASCII NAMES
- ASCII NAMES MAY BE UP TO 12 CHARACTERS LONG

JOB DIRECTORY

ASCII NAME	OBJECT TOKEN
'INTE6TASK'	8C58
'MBX1'	945C
•	•
•	•
•	•

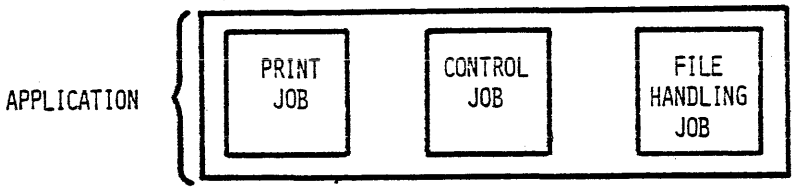
CATALOG/LOOKUP PROCESS

- OBJECTS ARE CATALOGED UNDER A USER GIVEN NAME BY THE TASK THAT CREATED THE OBJECT.
- TASKS WHICH KNOW THE NAME OF THE OBJECT THEN LOOK UP AN OBJECT'S TOKEN IN THE DIRECTORY.

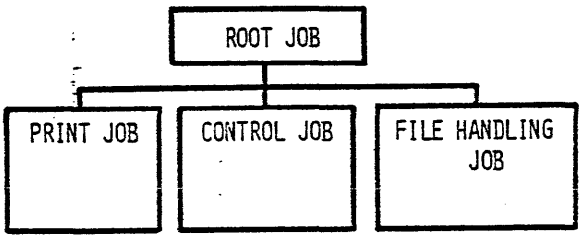


JOBS

- AN APPLICATION CAN CONSIST OF ONE OR MORE JOBS

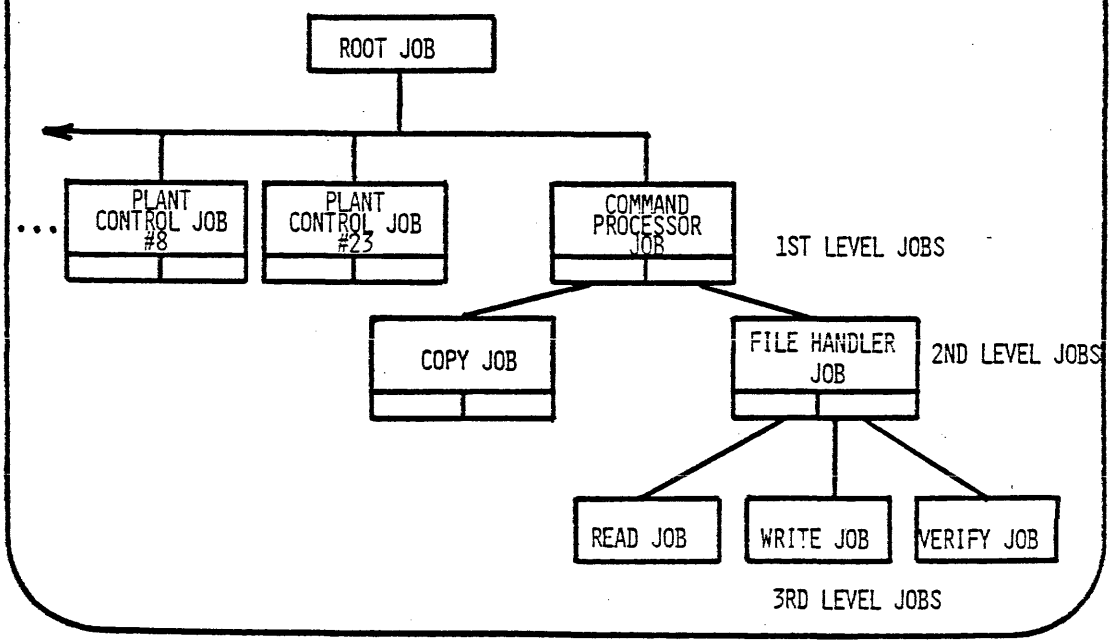


- AN APPLICATION'S JOBS FORM A FAMILY TREE
  - THE ROOT JOB IS PROVIDED BY THE iRMX 86 NUCLEUS
  - ALL OTHER JOBS ARE DESCENDENTS OF THE ROOT JOB

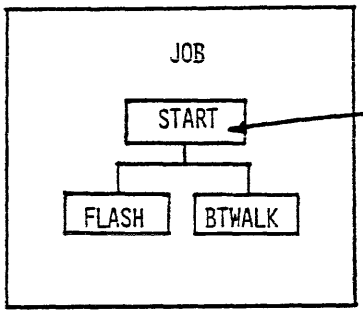


JOB TREES

- DEPENDING ON SYSTEM DESIGN, JOBS MAY BE SET UP IN MANY CONFIGURATIONS



A JOB MUST HAVE AN INITIALIZATION TASK



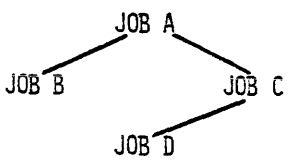
START TASK IS AN INITIALIZATION TASK FOR THIS JOB

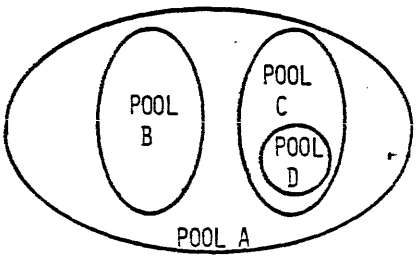
AN INITIALIZATION TASK:

- IS PLACED ON THE READY LIST WHEN THE JOB IS CREATED
- IS THE "ROOT TASK" FOR THE JOB (CREATES 1ST LEVEL TASKS IN JOB)
- CONTAINS AN RQENDINIT$TASK SYSTEM CALL

MEMORY POOLS

- CREATED OBJECTS (EXCEPT TASK$CODE) ARE ALLOCATED FROM MEMORY POOLS
- EACH JOB CONTAINS A MEMORY POOL WHICH WAS ALLOCATED FROM ITS PARENT'S POOL
- THERE IS A TREE-STRUCTURE HIERARCHY OF MEMORY POOLS EQUIVALENT TO HIERARCHY OF JOBS
- MEMORY THAT A JOB BORROWS FROM ITS PARENT REMAINS IN THE PARENT POOL





PARAMETER OBJECT TOKEN

- WHEN A TASK CREATES A JOB, IT CAN ALSO PASS A SINGLE TOKEN AS A PARAMETER TO THE NEWLY CREATED JOB

```
JOB$TOKEN = RQ$CREATE$JOB(..., PARAMETER$OBJECT$TOKEN, ...);
```

FOR EXAMPLE:

IN ORDER TO CATALOG IN A PARENT'S DIRECTORY, A TASK MUST KNOW THE TOKEN OF THE PARENT JOB. THE PARENT JOB TOKEN COULD BE PASSED IN THE RQ$CREATE$JOB CALL.

TOKENS AVAILABLE TO TASKS

THE RQGETTASK$TOKENS SYSTEM CALL MAKES TOKENS AVAILABLE TO THE CALLING TASK.

```
REQUEST$TOKEN = RQ$GET$TASK$TOKENS(SELECTION, EXCEPT$PTR);
```

THE TOKENS COME IN FOUR FLAVORS:

SELECTION
A BYTE INDICATING OBJECT TYPE OF REQUESTED TOKEN
= 0, THE CALLING TASK
= 1, THE CALLING TASK'S JOB
= 2, THE PARAMETER OBJECT OF CALLING TASK'S JOB
= 3, THE ROOT JOB

FOR EXAMPLE:

A JOB'S PARAMETER OBJECT TOKEN CAN BE OBTAINED BY A TASK IN THE CHILD JOB IF SELECTION = 2.

ADVANCED TOPICS ON

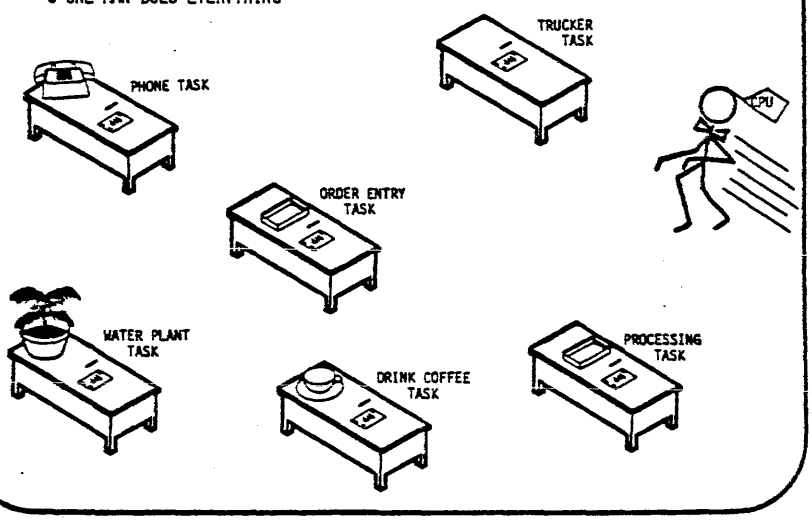
INTERTASK COMMUNICATION

- SEMAPHORES

REVISIT ONE MAN WIDGET MFG. CO.

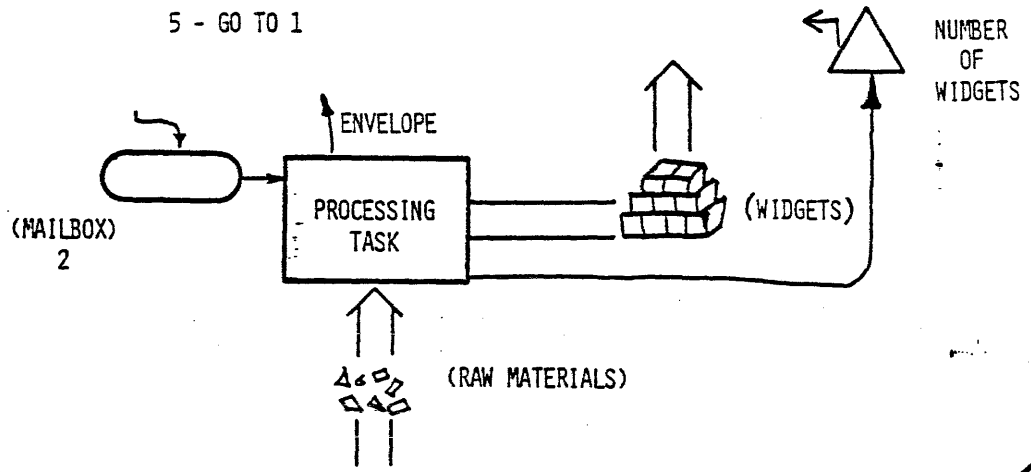
ONE MAN WIDGET MANUFACTURING CO.

- ONE MAN DOES EVERYTHING



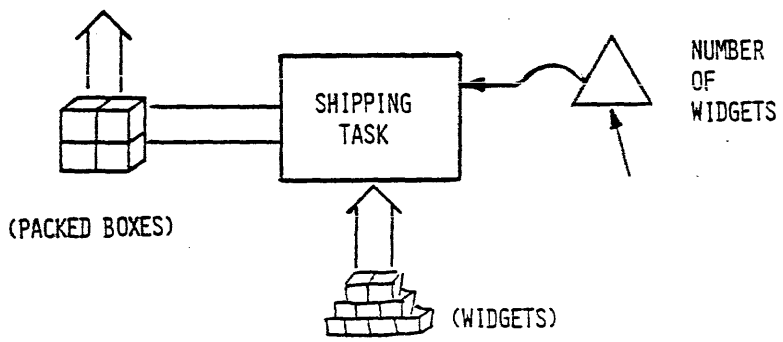
PROCESSING TASK

1 - WAIT FOR AN ENVELOPE IN MAILBOX 2
2 - GET RAW MATERIALS AND ASSEMBLE ONE WIDGET
3 - INCREMENT NUMBER OF WIDGETS ASSEMBLED BY ONE
4 - RETURN ENVELOPE TO RACK
5 - GO TO 1



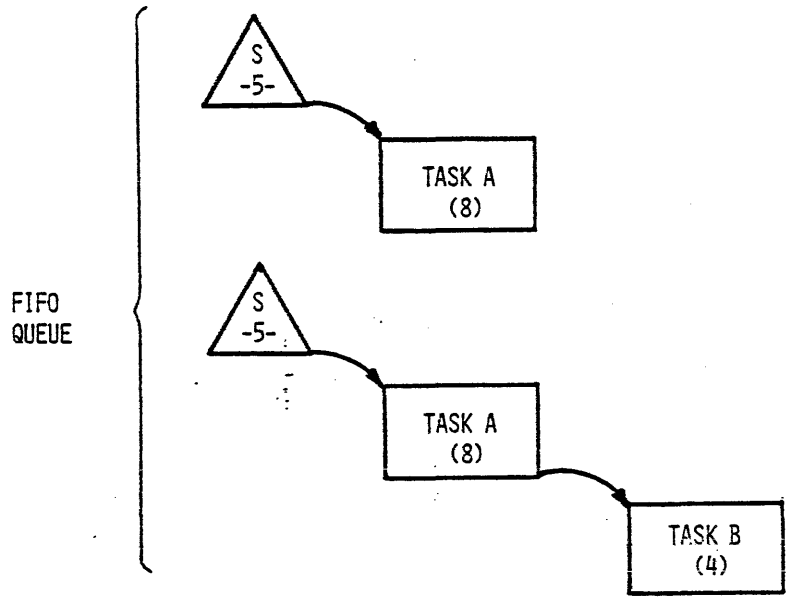
SHIPPING TASK

1 - WAIT FOR AT LEAST 5 WIDGETS ASSEMBLED
2 - DECREMENT NUMBER OF WIDGETS ASSEMBLED BY 5
3 - PACK INTO BOXES
4 - CALL PARCEL SERVICE COMPANY
5 - GO TO 1



SEMAPHORE QUEUE

A SEMAPHORE HAS A QUEUE OF TASKS WHICH CAN BE FIFO OR PRIORITY BASED.



TASK B CANNOT BE SATISFIED UNTIL AFTER TASK A HAS LEFT THE QUEUE.

SYSTEM CALLS FOR SEMAPHORES

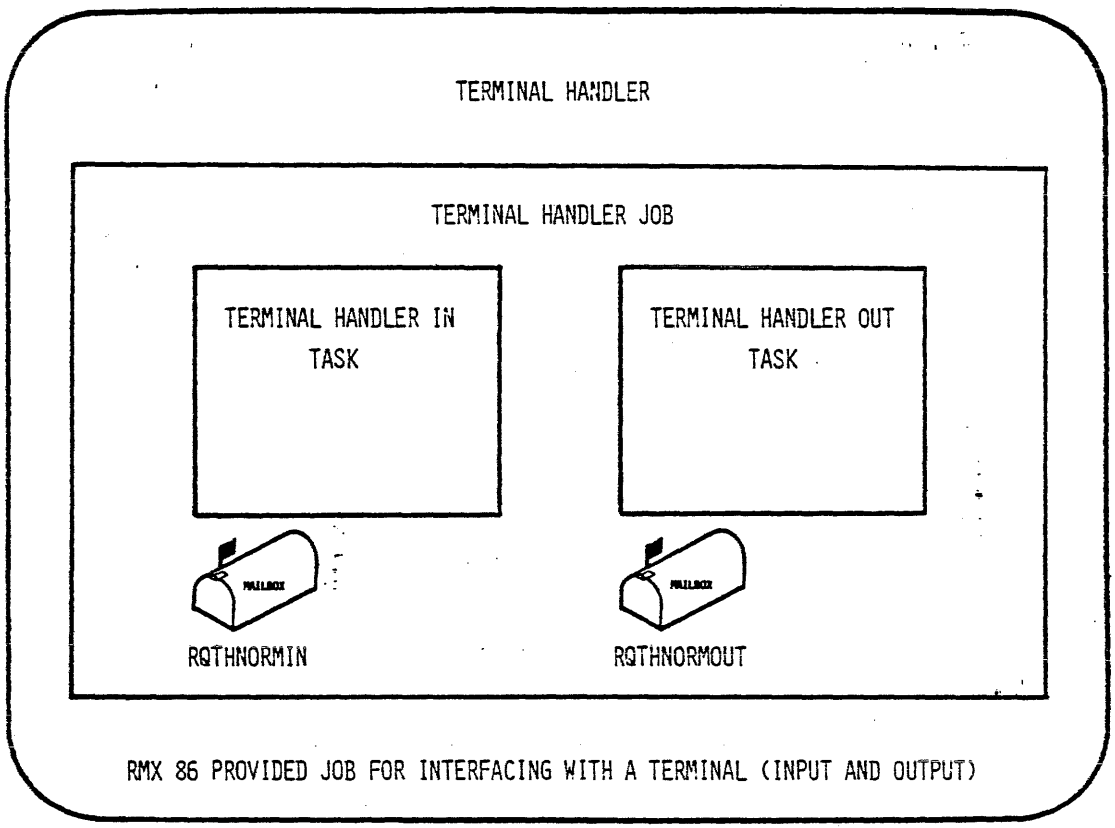
RQ$CREATESEMAPHORE - CREATES A SEMAPHORE AND RETURNS A TOKEN FOR IT

RQ$DELETE$SEMAPHORE - DELETES A SEMAPHORE FROM THE SYSTEM

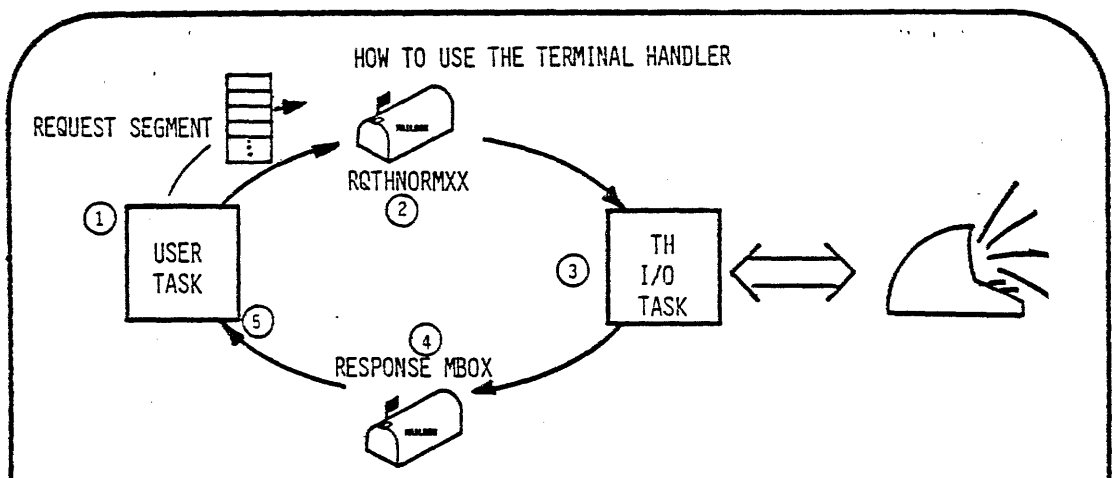
RQ$SEND$UNITS - ADDS A SPECIFIC NUMBER OF UNITS TO THE COUNT OF A SEMAPHORE

RQ$RECEIVE$UNITS - ASKS FOR A SPECIFIC NUMBER OF UNITS FROM A SEMAPHORE

- REFER TO NUCLEUS REFERENCE MANUAL FOR DETAILS ON PARAMETERS



RMX 86 PROVIDED JOB FOR INTERFACING WITH A TERMINAL (INPUT AND OUTPUT)



1. USER CREATES, INITIALIZES AND SENDS A SEGMENT TO THE APPROPRIATE TERMINAL HANDLER MAILBOX (RQTHNORMIN OR RQTHNORMOUT)
2. THE SEGMENT IS QUEUED AT THE MAILBOX
3. THE TERMINAL HANDLER PROCESSES THE REQUEST
   - THE REQUEST MAY BE READ OR WRITE
4. AFTER PROCESSING THE REQUEST, THE TERMINAL HANDLER RETURNS THE SEGMENT TO THE USER, VIA A RESPONSE MAILBOX
5. THE USER TASK WAITS AT THE RESPONSE MAILBOX

OUTPUT TO THE TERMINAL

- THE TASK SENDS AN OUTPUT REQUEST MESSAGE TO THE TERMINAL HANDLER'S MAILBOX 'RQTHNORMOUT'
- OUTPUT IS SENT BY THE TERMINAL HANDLER TO THE TERMINAL, ONE CHARACTER AT A TIME (A CARRIAGE RETURN, ØDH, IS ADDED TO THE OUTPUT WHEN A LINEFEED, ØAH, IS SEEN)
- THE TASK CAN WAIT AT ITS RESPONSE MAILBOX FOR SUCCESS OF THE OUTPUT ACTION
- IF NO RESPONSE MAILBOX IS GIVEN, IN THE RQSENDMESSAGE SYSTEM CALL, THE SEGMENT IS DELETED BY THE TERMINAL HANDLER

PLM AND THE REQUEST MESSAGE

```
.
.
/* DEFINE MESSAGE AND BASE IT */
DECLARE TH$SEG$TOKEN WORD;
DECLARE TH$REQ$MSG$PTR POINTER;
DECLARE TH$REQ$MSG$OVL STRUCTURE(OFFSET WORD, BASE WORD)
                                aTH$REQ$MSG$PTR);
DECLARE TH$REQ$MSG BASED TH$REQMSG$PTR STRUCTURE (FUNCTION WORD,
                                                  COUNT WORD,
                                                  EXCEPTION$CODE WORD,
                                                  ACTUAL WORD,
                                                  BUFFER (132) BYTE);
.
.
.
/* CREATE SEGMENT FOR MESSAGE */
THE$SEG$TOKEN=RQ$CREATE$SEGMENT(140, aSTATUS);
TH$REQ$MSG$P.BASE=TH$SEG$TOKEN;
TH$REQ$MSG$OVL.OFFSET=0
/* SET MESSAGE VALUES */
THE$REQ$MSG.FUNCTION=F$READ;
TH$REQ$MSG.COUNT=132
```

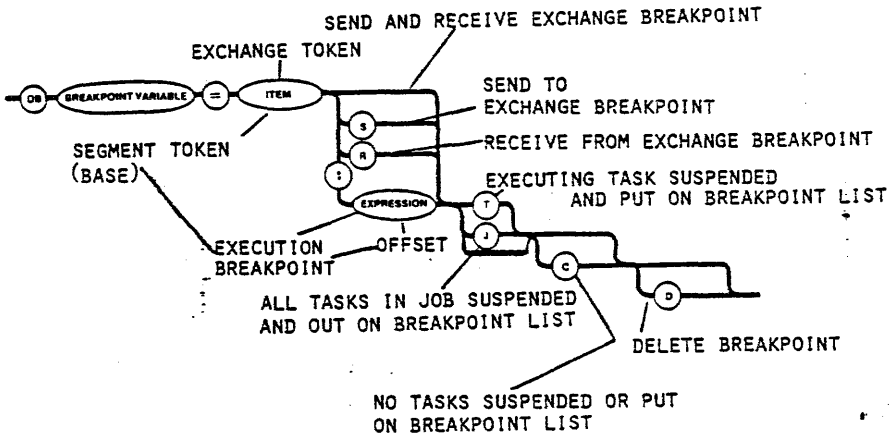
THE DEBUGGER

DEBUGGER

- REAL TIME DEBUGGING CAPABILITY
- CAN VIEW RMX NUCLEUS DATA STRUCTURES
- VIEW/CHANGE VARIABLES AND BREAKPOINTS
- DEBUGGER INVOKED BY A CONTROL-D ENTERED AT THE TERMINAL
  - PROMPT IS '*'

SET A BREAKPOINT

*DB .BRKPT1 = 1011T



THE BREAKPOINT LIST

TO VIEW THE BREAKPOINT LIST:

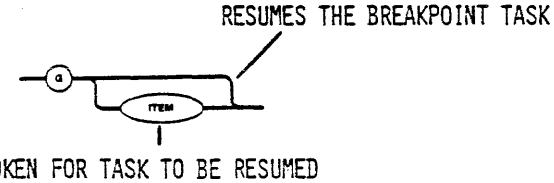
*BL

BL = 0104J/0076TX 0104J/0092TE

TASK INCURRED AN EXECUTION BREAKPOINT	TASK INCURRED IN AN EXCHANGE BREAKPOINT

- THE G COMMAND RESUMES A TASK AND REMOVES IT FORM THE BREAKPOINT LIST

EXAMPLE: *G 0092

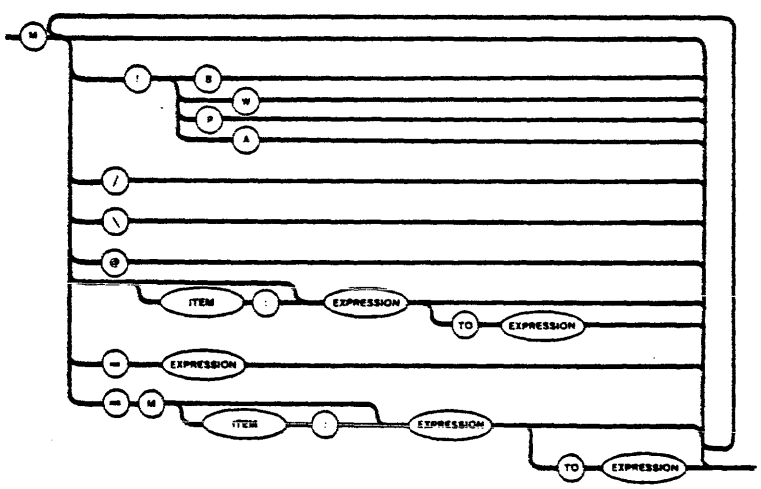


EXAMPLE OF OUTPUT FROM THE I COMMAND

```
-IJ 2FEB 0
                    ----- iRMX 86  JOB  REPORT -----
JOB TOKEN                 2FEB        PARENT JOB                ROOT
POOL MAXIMUM              FFFF        POOL MINIMUM              0040
CURRENT ALLOCATED         0B62        CURRENT UNALLOCATED       01BF
CURRENT # OBJECTS         0004        MAXIMUM # OBJECTS         FFFF
CURRENT # TASKS           0001        MAXIMUM # TASKS           FFFF
CURRENT # CHILDREN JOBS   0003        DELETION PENDING          NO
EXCEPTION MODE            0000        EXCEPTION HANDLER         2420:0235
MAXIMUM PRIORITY          0000
NAME(S)   NONE FOUND
                    -----OBJECT  DIRECTORY-----
MAXIMUM SIZE              000A        VALID ENTRIES             0002
NAME            TOKEN     NAME            TOKEN     NAME            TOKEN
RQTHNORMIN       2F84     RQTHNORMOUT      2F20

-IT 2F69
                    ----- iRMX 86  TASK  REPORT -----
TASK TOKEN                2F69        CONTAINING JOB            2FBC
STACK SEGMENT BASE        2F40        STACK SEGMENT OFFSET      01DA
STACK SEGMENT SIZE        0200        STACK SEGMENT LEFT        0134
CODE SEGMENT BASE         0800        DATA SEGMENT BASE         0DA9
INSTRUCTION POINTER       2E09        TASK STATE                READY
STATIC PRIORITY           0082        DYNAMIC PRIORITY          0082
SUSPENSION DEPTH          0000        SLEEP UNITS REQUESTED     FFFF
EXCEPTION MODE            0000        EXCEPTION HANDLER         2420:0235
NAME(S)   NONE FOUND
```

EXAMINING OR MODIFYING MEMORY - THE M COMMAND



SUMMARY

- BREAKPOINT RELATED COMMANDS

  B - VIEW BREAKPOINT PARAMETERS, BREAKPOINT LIST, AND BREAKPOINT TASK INFORMATION
  BL - VIEW BREAKPOINT LIST
  BT - INQUIRE ABOUT BREAKPOINT TASK
  DB - DEFINE A BREAKPOINT
  G - REMOVE A TASK FROM THE BREAKPOINT LIST
  R - VIEW/CHANGE BREAKPOINT TASKS REGISTERS
  Z - DELETE A BREAKPOINT
  . VARIABLE - CHANGE/EXAMINE BREAKPOINT NAMED

- MEMORY LOCATION RELATED COMMANDS

  D - DEFINE NUMERIC VALUE
  I - EXAMINE SYSTEM OBJECTS
  L - LIST NUMERIC VARIABLES
  M - EXAMINE MODIFY MEMORY
  Q - EXIT THE DEBUGGER
  V - VIEW SYSTEM LISTS
  .VARIABLE - CHANGES VALUE OF VARIABLE NAMED

RMX 86 TEST

RMX 86 TEST (OPEN BOOK)

② CONTINUED

```
DECLARE (THIN, RSPMBX,ROOTKN,SEGTKN,STATUS) WORD;
RSPMBX = RQCREATEMAILBOX (_____, _____);
ROOTKN = RQGETTASKTOKENS (_____, _____);
THIN = RQLOOKUPOBJECT (_____, _____, _____, _____);
SEGTKN = RQCREATESEGMENT (_____, _____);
```

RMX 86 TEST (OPEN BOOK)

③ WHAT IS THE MOST "ECONOMICAL" WAY TO SYNCHRONIZE TWO TASKS?

④ CAN A JOB BE DELETED IF IT CONTAINS AN INTERRUPT TASK?

RMX 86 TEST (OPEN BOOK)

⑩ FREAD = _____

FWRITE = _____

⑪ WHAT DO YOU NEED TO DO, TO MAKE THE TERMINAL HANDLER, DELETE THE SEGMENT AFTER IT HAS OUTPUTTED THE MESSAGE?

RMX 86 TEST (OPEN BOOK)

⑫ GIVEN: THE FOLLOWING INTERRUPT HANDLER + TASK

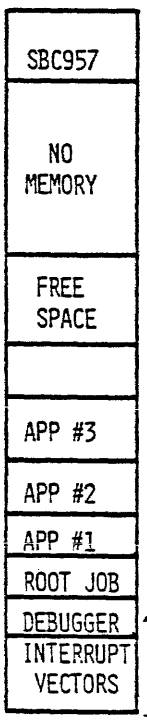
```
INTHND:PROCEDURE INTERRUPT X;
   ~~~~~~
   ~~~~~~
RQSIGNALINTERRUPT(LEVEL 3,@STATUS)
END;
```

```
INTTASK:PROCEDURE PUBLIC;
    CALL RQSETINTERRUPT(LEVEL$3,,...);
DOFOREVER;
    CALL RQWAITINTERRUPT(LEVEL$3,,...);
        /*PROCESS*/
END;
END;
```

RMX 86 DISKETTES

- THE O.S. COMES FROM THE FACTORY IN SEVERAL DISKETTES (ISIS FORMAT, SINGLE OR DOUBLE DENSITY).
- EACH DISKETTE CONTAINS A SUBSYSTEM, CONFIGURATION FILE(S), LIBRARIES, AND A SUBMIT FILE.
- THE CONFIGURATION FILES ARE WRITTEN AS ASSEMBLY LANGUAGE MACROS.
- DESIRED FEATURES OF A SUBSYSTEM CAN BE SELECTED BY MODIFYING THE CONFIGURATION FILES (EG, NUCLEUS CAN BE CONFIGURED FROM 12K TO 26K OF CODE DEPENDING ON NUMBER OF FEATURES REQUIRED).
- THE CONFIGURATION FILES HAVE THE EXTENSION .A86, THEY ARE MODIFIED THROUGH THE TEXT EDITOR
  (EG, CREDIT :F1: NTABLE.A86).

SYSTEM LAYOUT



- LAY OUT THE SELECTED SUBSYSTEMS
- ALLOW SPACE FOR THE ROOT JOB (BETWEEN 500 TO 1000 BYTES)
- LAY OUT EACH APPLICATION SOFTWARE JOB IN TURN
- APPLICATION JOBS ARE THE MOST VOLATILE! LAY THESE OUT LAST.

THE DEBUGGER JOB CONTAINS THE TERMINAL HANDLER JOB

NDEVCF.A86

- THE MASTER_PIC AND TIMER MACROS MUST ALWAYS BE INVOKED
- IF SLAVES ARE CONFIGURED IN THE SYSTEM, THE MASTER_PIC MACRO MUST BE INVOKED PRIOR TO ANY SLAVE_PIC MACRO INVOCATIONS
- INVOCATIONS OF THE SLAVE_PIC AND NDP_SUPPORT MACROS ARE OPTIONAL

TO CONFIGURE A DEVICE INTO THE iRMX 86 SYSTEM AND TO DEFINE ITS ATTRIBUTES, INVOKE THE FOLLOWING MACROS WITH THE DESIRED PARAMETERS.

NUCLEUS DEVICE CONFIGURATION TABLE

```
%MASTER_PIC(8259A,0C0H, 0,0,)
;SLAVE_PIC( SLAVE_TYPE, BASE_PORT, EDGE_VS_LEVEL, MASTER_LEVEL)
%TIMER(8253,0D0H, 28H,12288)
%NDP_SUPPORT( 08H)
```

- REFER TO CONFIGURATION MANUAL FOR FURTHER DETAILS.

LOCATE ADDRESSES

- THE LOCATE UTILITY PROGRAM GENERATES A MAP LIST FILE CALLED NUCLUS.MP2
- EXAMING THE MAP WE OBTAIN THE ENDING ADDRESS OF THE NUCLEUS

MEMORY MAP OF MODULE NBEGIN

SEGMENT MAP

START	STOP	LENGTH	ALIGN	NAME	CLASS	OVERLAY
00000H	003FFH	00400H	A	(ABSOLUTE		
01040H	07077H	6038H	W	CODE	CODE	
07078H	07091H	001AH	W	OBJ_SEG	CODE	
92H	0709BH	000AH	W	JOB_SEG	CODE	
0709CH	070AFH	0014H	W	TASK_SEG	CODE	
070B0H	070B7H	0008H	W	MB_SEG	CODE	
070B8H	070BFH	0008H	W	SEM_SEG	CODE	
070C0H	070C9H	000AH	W	REG_SEG	CODE	
070CAH	070D7H	000EH	W	FS_SEG	CODE	
070D8H	070F1H	001AH	W	INT_SEG	CODE	
	.					
	.					
	.					
07190H	07190H	0000H	W	STACK	STACK	
07190H	07190H	0000H	W	MEMORY	MEMORY	

ROOT JOB CONFIGURATION

- THE CONFIGURATION FILE FOR THE ROOT JOB IS NOT PROVIDED IN THE SYSTEM DISKETTE
- CONFIGURATION FILE IS A SINGLE SOURCE FILE WHICH DESCRIBES:
  - EACH FIRST-LEVEL JOB TO BE CREATED
  - THE APPLICATION SYSTEM ADDRESS BLOCKS
  - THE APPLICATION SYSTEM AS GLOBAL ATTRIBUTES
- THE CONFIGURATION INFORMATION IS PROVIDED BY ASSEMBLY MACRO CALLS
  - %JOB — DEFINES JOB PARAMETERS FOR EACH FIRST LEVEL APPLICATION JOB
  - %SAB — DEFINES MEMORY NOT TO BE ASSIGNED TO THE FREE SPACE MANAGER AT INITIALIZATION
  - %SYSTEM — DEFINES SYSTEM PARAMETERS FOR THE SYSTEM CONFIGURATION

% JOB

MACRO CALL: __JOB (DEFINES FIRST-LEVEL JOBS)_____
NUMBER OF CALLS REQUIRED: ONE FOR EACH FIRST-LEVEL JOB
CONFIGURATION FILE NAME: _____

FORMAT:

	PARAMETER	TYPE	SUGGESTED DEFAULT	VALUE
%JOB	(DIRECTORY_SIZE,	WORD		
	POOL_MIN,	WORD		
	POOL_MAX,	WORD	(OFFFFH)	
	MAX_OBJECTS,	WORD		
	MAX_JOB_PRIORITY,	BYTE		
	EXCEPTION_HANDLER_ENTRY,	ADDR	(0:0)	
	EXCEPTION_HANDLER_MODE,	BYTE	(1)	
	JOB_FLAGS,	WORD	(0)	
	INIT_TASK_PRIORITY,	BYTE	(0)	
	INIT_TASK_ENTRY,	ADDR		
	DATA_SEGMENT_BASE,	BASE	(0)	
	STACK_POINTER,	ADDR	(0:0)	
	STACK_SIZE,	WORD	(512)	
	TASK_FLAGS)	WORD	(0)	

THE %SYSTEM MACRO

- DEFINES GLOBAL APPLICATION SYSTEM PARAMETERS

MACRO CALL: %SYSTEM (SYSTEM PARAMETERS)

NUMBER OF CALLS REQUIRED: EXACTLY ONE

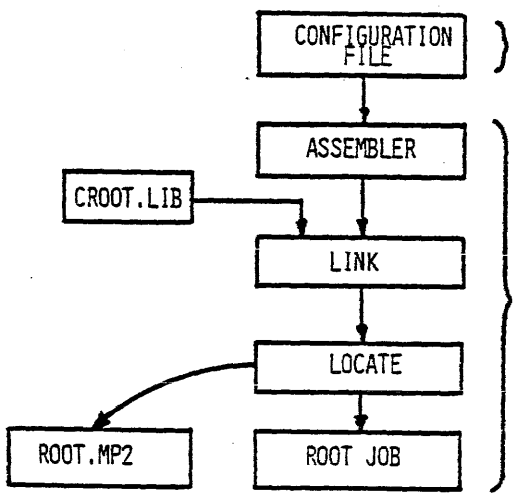
CONFIGURATION FILE NAME _____

FORMAT:

	PARAMETER	TYPE	SUGGESTED DEFAULT	VALUE
%SYSTEM	(NUCLEUS_ENTRY,	BASE		____
	ROD_SIZE,	WORD		____
	MIN_TRANS_SIZE,	WORD	(64)	____
	DEBUGGER,	SEE NOTE 1	(A)	____
	DEFAULT_E_H_PROVIDED,	SEE NOTE 2	(N)	____
	MODE)	WORD		

STEPS IN BUILDING THE ROOT JOB

1) CREATE A CONFIGURATION FILE
2) ASSEMBLE THE CONFIGURATION FILE
3) LINK AND LOCATE THE ROOT JOB



CREDIT :F1: ROOT.A86

SUBMIT :F1:CROOT(FILE,DATE,LOC_ADDR)

WHAT/WHY UDI?

UNIVERSAL DEVELOPMENT INTERFACE

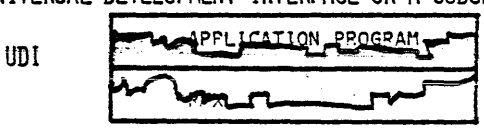
UDI IS A SPECIFICATION OF A SET OF PROCEDURE CALLS THAT ARE USED TO REQUEST OPERATING SYSTEM FUNCTIONS.

THE KINDS OF FUNCTIONS THAT ARE AVAILABLE THROUGH UDI PROCEDURE CALLS INCLUDE:

- CREATING AND BREAKING CONNECTIONS TO DATA FILES
- OPENING, READING, SEEKING, WRITING, AND CLOSING DATA FILES
- CONTROLLING PROGRAM EXECUTION
- CONTROLLING MEMORY ALLOCATIONS
- HANDLING SYSTEM EXCEPTION CONDITIONS
- CONTROLLING THE PROCESSING OF CONSOLE INPUT & PARSING COMMAND LINES
- FETCHING THE CURRENT DATE AND TIME

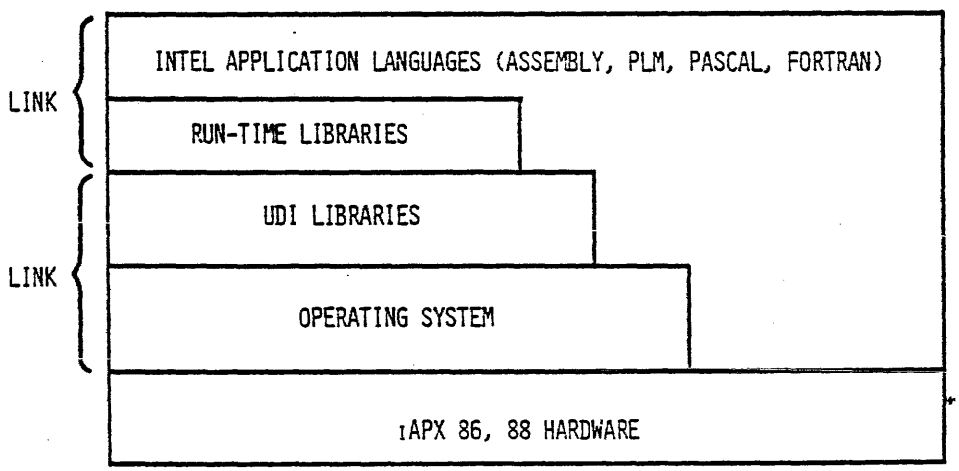
FUNCTIONS ARE IMPLEMENTED BY MODULES THAT TRANSLATE FROM THE UDI STANDARD TO THE ACTUAL OPERATING SYSTEM CALLS

EACH INTEL OPERATING SYSTEM FOR THE iAPX 86,88 FAMILY PROVIDES A UNIVERSAL DEVELOPMENT INTERFACE OR A SUBSET THEREOF.



LIBRARIES

THE iRMX 86 OPERATING SYSTEM SUPPORTS UDI BY PROVIDING UDI INTERFACE LIBRARIES.



THE iRMX OPERATING SYSTEM CONSISTS OF A NUMBER OF SUBSYSTEMS

RMX LAYERS	DESCRIPTION
NUCLEUS	THE CORE OF THE iRMX 86 OPERATING SYSTEM AND IS REQUIRED FOR EVERY APPLICATION SYSTEM
TERMINAL HANDLER	PROVIDES A REAL-TIME INTERFACE BETWEEN YOUR TERMINAL AND OTHER SOFTWARE.
BASIC I/O SYSTEM	PROVIDES ASYNCHRONOUS FILE ACCESS CAPABILITIES
EXTENDED I/O SYSTEM	PROVIDES HIGH LEVEL, SYNCHRONOUS FILE ACCESS CAPABILITIES
APPLICATION LOADER	PROVIDES THE CAPABILITY TO LOAD OBJECT FILES INTO MEMORY FROM DISK
HUMAN INTERFACE	PROVIDES AN INTERACTIVE INTERFACE BETWEEN A USER AND SOFTWARE

UDI CALLS AND iRMX 86 SYSTEM CALLS

UDI CALLS	iRMX 86 SYSTEM CALLS	SUBSYSTEMS
DQ$ALLOCATE	RQ$CREATE$SEGMENT	NUCLEUS
DQ$ATTACH	RQSATTACH$FILE	EXTENDED I/O SYSTEM
DQ$CHANGE$EXTENSION	(NONE)	(NONE)
DQ$CLOSE	RQSCLOSE	EXTENDED I/O SYSTEM
DQ$CREATE	RQSCREATE$FILE RQ$SGETFILE$STATUS	EXTENDED I/O SYSTEM
DQ$DECODE$EXCEPTION	RQCFORMAT$EXCEPTION	HUMAN INTERFACE
DQ$DELETE	RQ$DELETE$FILE	EXTENDED I/O SYSTEM
DQ$DETACH	RQSDELETE$CONNECTION RQ$S$CLOSE	EXTENDED I/O SYSTEM
DQ$FREE	RQ$DELETE$SEGMENT	NUCLEUS

ERROR REPORTING

UDI PROCEDURES RETURN A CONDITION CODE THAT INDICATES THE RESULTS OF EXECUTING A UDI PROCEDURE.

- YOU MUST CHECK THE CONDITION CODE AFTER EACH UDI CALL TO ENSURE PROPER RESULTS

TABLE 6-2. iRMX 86 EXCEPTION CODES AND MNEMONICS

HEX CODE	MNEMONIC	HEX CODE	MNEMONIC
0000	E$OK	0065	E$EOF
0001	E$TIME	0066	E$FIXUP
0002	E$MEM	0067	ENOLOADER$MEM

•
•
•

(SEE COMPLETE LISTING IN RUN TIME SUPPORT MANUAL)

OTHER UDI FACTS

INTERRUPT HANDLING

PROGRAMS THAT RUN UNDER THE iRMX 86 OPERATING SYSTEM SHOULD USE iRMX 86 INTERRUPT MANAGEMENT TECHNIQUES TO HANDLE INTERRUPTS.

- THE UDI LIBRARIES DO NOT INCLUDE INTERRUPT MANAGEMENT.

REENTRANCY

UDI LIBRARIES ARE FULLY REENTRANT WITH THE FOLLOWING RESTRICTIONS:

- EACH JOB MUST HAVE ITS OWN COPY OF THE UDI LIBRARIES.
- YOU CAN HAVE ONLY ONE COPY OF THE UDI LIBRARIES WITHIN A SINGLE JOB.

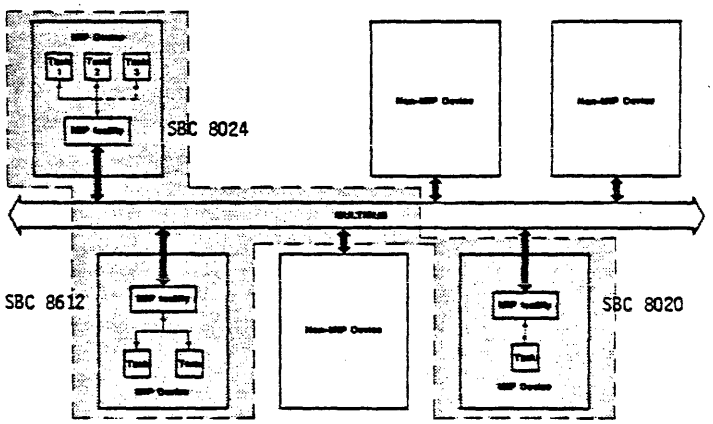
MULTITASKING

- THE UDI LIBRARIES ARE FULLY COMPATIBLE WITH A MULTITASKING ENVIRONMENT. HOWEVER, THERE ARE NO UDI CALLS TO CREATE AND DELETE TASKS.

MIP and iMMX 800

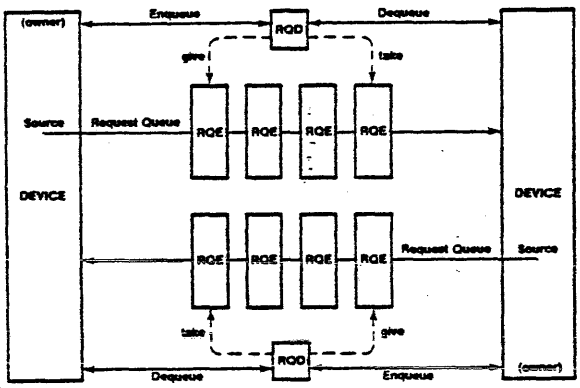
WHAT IS MIP?

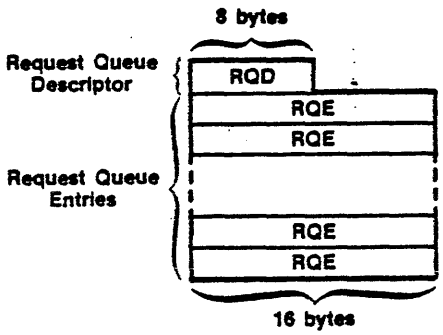
- THE MULTIBUS INTERPROCESSOR PROTOCOL (MIP) IS A SPECIFICATION FOR A SET OF MECHANISMS AND PROTOCOLS.
- PROVIDES AN EXCHANGE OF DATA AMONG TASKS EXECUTING ON VARIOUS SINGLE-BOARD COMPUTERS.



CHANNELS

- COMMUNICATION BETWEEN DEVICES IS IMPLEMENTED USING CHANNELS
  - A CHANNEL CONSISTS OF A PAIR OF QUEUES
  - ONE CHANNEL MUST BE DEFINED FOR EACH DEVICE PAIR WHICH WILL COMMUNICATE WITH EACH OTHER
- A CHANNEL MUST RESIDE IN A MEMORY SEGMENT ACCESSIBLE BY BOTH DEVICES WHICH USE THAT CHANNEL
  - GLOBAL MEMORY
  - DUAL PORT MEMORY



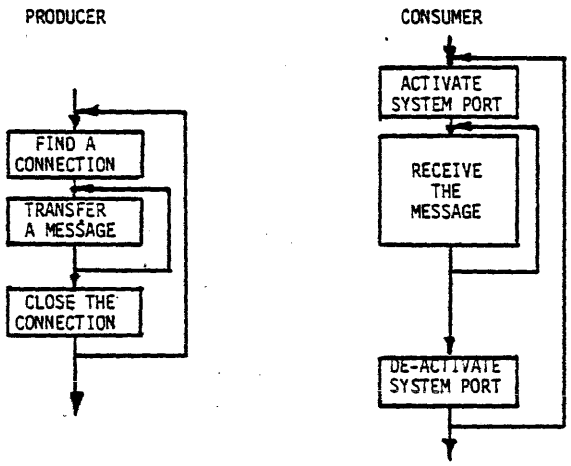


iMMX 800

- iMMX 800 IS THE IMPLEMENTATION OF THE MIP SPECIFICATION
- COMES IN THREE VERSIONS
  - OPERATION UNDER iRMX 80 NUCLEUS
    - *iMMX 800/80
  - OPERATION UNDER iRMX 88 NUCLEUS
    - *iMMX 800/880 FOR NON-MEGABYTE SUPPORT
    - *iMMX 800/881 FOR MEGABYTE SUPPORT
  - OPERATION UNDER iRMX 86 NUCLEUS
    - *iMMX 800/86
- ALL THREE VERSIONS PRESENT IDENTICAL USER INTERFACES
- CONSISTS OF THREE PARTS
  - MESSAGE MANAGER
    - *PROVIDES INTERFACE TO USER TASKS (SEND, RECEIVE)
  - PARTITIONED MEMORY MANAGER (PMM)
    - *MANAGES MEMORY POOLS
  - SIGNAL MANAGER
    - *PROVIDES INTERFACE TO OTHER DEVICES

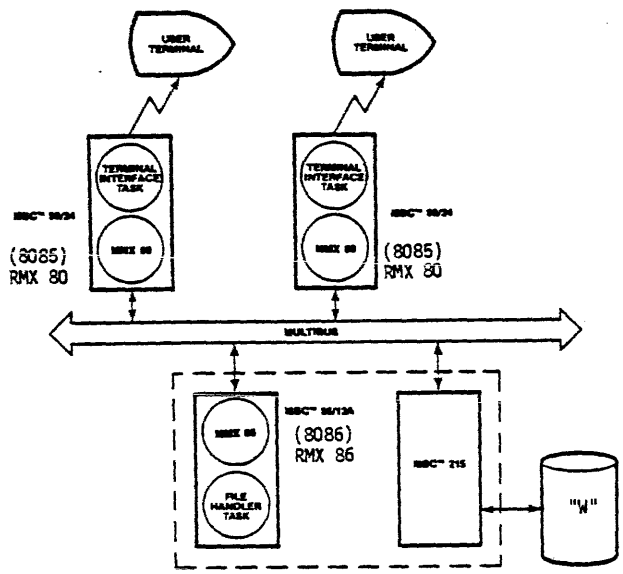
MESSAGE MANAGER

- FUNCTIONALITY FROM USER VIEWPOINT



DATA BASE APPLICATION EXAMPLE

- 2 OPERATORS - 2 TERMINALS ACCESSING DATA FILES
- THE TERMINALS CONTROLLED BY RMX 80
- THE DATA BASE ("WINCHESTER") CONTROLLED BY RMX 86 BASIC I/O SYSTEM



THE

80130

REVIEW/QUIZ

1) WHAT IS A CONTEXT SAVE?

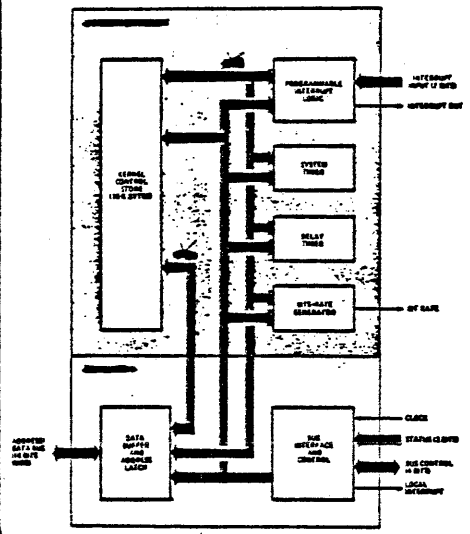
2) WHAT ARE DIFFERENCES BETWEEN CALL RQSLEEP AND CALL TIME?

3) WHAT ARE THE TWO CONDITIONS IMPOSED ON A TASK BEFORE IT BECOMES THE RUNNING TASK?

4) WRITE THE COMMAND (USING THE RMX86 DEBUGGER) TO BREAK POINT A TASK (9C4F) AT A SEMAPHORE (9D40) WHEN THE TASK RECEIVES 3 UNITS FROM THAT SEMAPHORE.

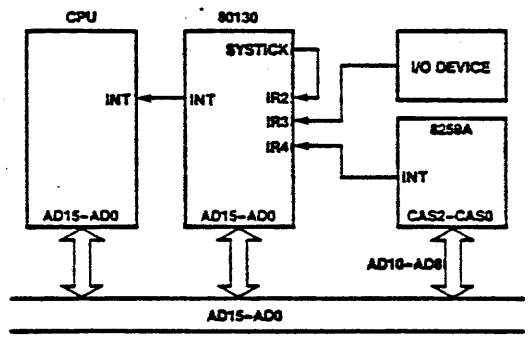
5) CAN I HAVE MORE THAN ONE TERMINAL HANDLER?

HARDWARE FEATURES



- 128K BIT CONTROL STORAGE
  - ZERO WAIT STATES AT 8MHZ (200ns ACCESS TIME)
  - ADDRESSABLE AS 16K X 8 BITS OR 8K X 16 BITS
- PROGRAMMABLE INTERRUPT CONTROLLER
  - COMPLETELY MANAGED BY PRIMITIVES
  - 8 INTERRUPT INPUTS INDIVIDUALLY MASKED BY THE O.S.
  - CAN EXPAND TO 57 INPUTS WITH CASCADED 8259-A'S
- 3 PROGRAMMABLE TIMERS: SYSTEM, DELAY, BAUD RATE GENERATION
  - SYSTEM TIMER TO A MINIMUM OF 1 msec
  - DELAY COUNTS DOWN SYSTEM TIMER INTERVALS (SIGNALS ON ZERO)
  - BAUD RATE GENERATION — 8254 OPERATES IN SQUARE WAVE MODE

OSF INTERRUPT CONTROLLER OPERATION



- ☐ Operation is similar to 8259A PIC
- ☐ One or more interrupt inputs are activated
- ☐ 80130 activates INT line to notify CPU of interrupt request
- ☐ CPU acknowledges interrupt with two interrupt acknowledge (INTA) cycles
- ☐ For external 8259As, 80130 drives cascade address (CAS2–CAS0) on AD10–AD8 during second INTA cycle
- ☐ An 8-bit interrupt vector is returned to the CPU by either the 80130 or by the selected slave 8259A during the second INTA cycle

PRIMITIVES FOR TASKS

- Create$task
  - — Creates a task and returns a token for it
- Delete$task
  - — Deletes a task that is not an interrupt task
- Suspend$task
  - — Increases a task's suspension depth by one. Suspends the task if it is not already suspended
- Resume$task
  - — Decreases a task's suspension depth by one. Resumes (unsuspends) the task if the suspension depth becomes zero
- Sleep
  - — Places the calling task in the asleep state for a specified amount of time
- Get$task$tokens
  - — Returns a token for either the calling task, the calling task's job, the parameter object of the calling task's job, or the root job
- Set$priority
  - — Changes the priority of a noninterrupt task

PRIMITIVES FOR MAILBOXES

- Create$mailbox
  - — Creates a mailbox and returns a token for it
- Delete$mailbox
  - — Deletes a mailbox from the system
- Send$message
  - — Sends an object to a mailbox
- Receive$message
  - — Sends the calling task to a mailbox for an object (the task has the option of waiting if no objects are present)

PRIMITIVES FOR JOBS

- Create$job
  - — Creates a job with a task and returns a token for the job

PRIMITIVES FOR ADDITIONAL SUPPORT

- Set$exception
  - — Assigns an exception handler to the calling task and sets the exception mode attributes
- Get$exception
  - — Returns to the calling task the current values of its exception handler and exception mode attributes
- Get$type
  - — Accepts a token for an object and returns the object's type code
- Disable$deletion
  - — Makes an object susceptible to ordinary deletion
- Set$O.S.$extension
  - — Enters or deletes the address of an entry or function procedure in the interrupt vector table
- Signal$exception
  - — Invoked by O.S. extensions to signal the occurrence of an exceptional condition

80130 INITIALIZATION AND CONFIGURATION CODE REQUIREMENTS

The complete set of iRMX 86 facilities can be added to an iAPX 86/30 or 88/30 system to meet extensive requirements

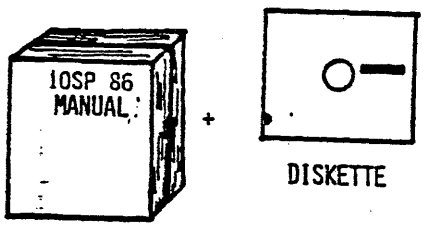
Using 80130 with BIOS of iRMX 86
- — 4.5K Bytes without parameter validation
- — 6.5K Bytes with parameter validation

Using 80130 with EIOS, human interface, or UDI
- — 8K Bytes without parameter validation
- — 10.5K Bytes with parameter validation

Note: This code must be adjacent to 80130's address space (The BIOS, EIOS, and HI can reside elsewhere!)

Also: The numbers do not include BIOS, EIOS, or HI code

80130 CONFIGURATION



IOSP SUPPORT PACKAGE

DISKETTE CONTENTS

CONFIGURATION FILES

- DEVICE CONFIGURATION TABLE
- PRIMITIVE/FEATURES CONFIGURATION TABLE

SUBMIT FILES

- ASSEMBLE, LINK, LOCATE CONFIGURATION FILES, APPLICATION CODE AND ROOT JOB

INTERFACE LIBRARIES

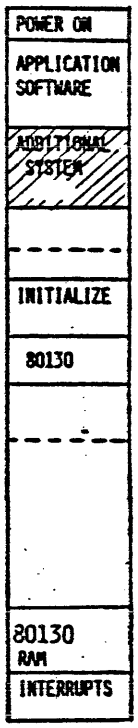
- INTERFACE FROM APPLICATION CODE CALLS TO PRIMITIVES
- COMPACT, MEDIUM AND LARGE LIBRARIES SUPPLIED

CODE LIBRARIES CONTAINING "FRONT ENDS" TO 80130 PRIMITIVES

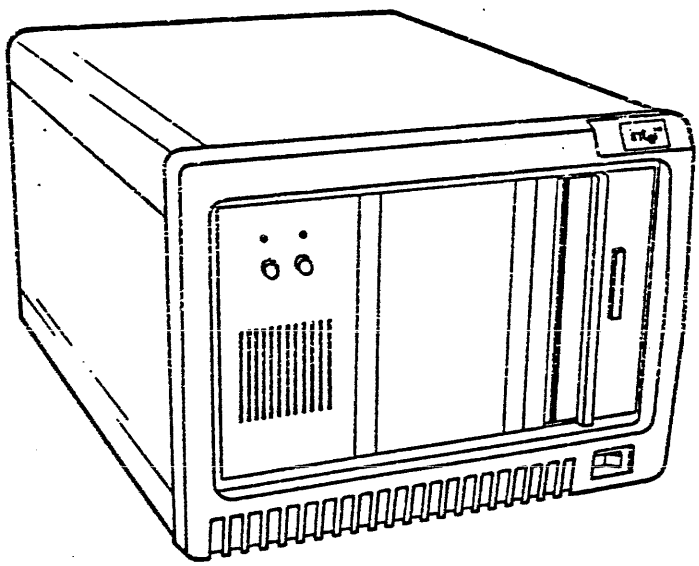
- INITIALIZATION CODE

GENERATE THE SYSTEM CONFIGURATION (ROOT JOB)

1) CREATE CONFIGURATION MODULE
   USING FOUR TYPES OF MACROS
   - %SAB - DEFINES ADDRESS BLOCKS NOT ASSIGNED TO FREE SPACE MANAGER
   - %JOB - DEFINES JOB PARAMETERS FOR EACH FIRST LEVEL JOB
   - %OSX - DEFINES BASE ADDRESS OF 80130
   - %SYSTEM - DEFINES SYSTEM WIDE PARAMETERS
2) ASSEMBLE CONFIGURATION MODULE TO CREATE ROOT JOB
3) LINK AND LOCATE ROOT JOB



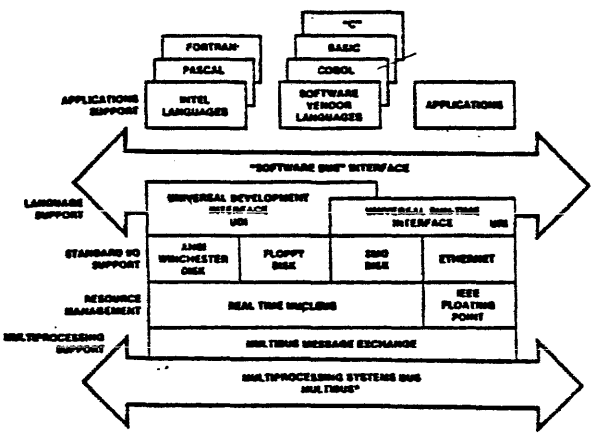
INTEL SYSTEM 86/330



86/330 SOFTWARE

- INTEL
  - LINK86
  - LOC86
  - OH86
  - DIAGNOSTICS
  - PLM86
  - ASM86
  - LIB86
  - EDIT86
  - SYSTEM DEBUG MONITOR
  - PASCAL86 (OPTION)
- THIRD PARTY
  - "C" WHITESMITH
  - MICROFOCUS COBOL
  - MICROSOFT COBOL (6182)
  - MICROSOFT BASIC INTERPRETER
  - MICROSOFT BASIC COMPILER (6182)

86/330 SOFTWARE



- **COMPLETE SUPPORT FOR THE UDI INTERFACE**
- **IMPROVED DEBUG CAPABILITIES**
- **LINE PRINTER DRIVER**
- **DISK VERIFY**
- **DISK BACKUP**
- **CONFIGURATION UNDER IRMX 86**

iRMX 86 RELEASE 5

BIOS

- PERFORMANCE ENHANCEMENTS
  - NEW STATUS INTERFACE-WAITFORIO
    -NO IORS
    -FASTER
  - OVERLAPPED SEEKS
  - UPDATE/TIMEOUT "CLOCKED"
  - "HOT SPOT" TUNING
- iSBC 215/iSBX 217 STREAMER TAPE DRIVER
- MULTITERMINAL SUPPORT
  - 534 DRIVER
  - 544 DRIVER
  - BOTH WITH CONFIGURABLE INTERRUPTS
- AUTO DENSITY RECOGNITION ON ATTACHMENT OF DEVICE!
  - 208 DRIVER
  - iSBX 218/iSBC 215 DRIVER

iRMX 86 RELEASE BIOS

TERMINAL DRIVER

- SUPPORTS 534 AND/OR ONBOARD USART
- PHYSICAL FILE INTERFACE
- FEATURES
  - ASCII CRT AND HARD COPY
  - TYPE AHEAD
  - LINE EDIT
  - TRANSPARENT MODE (ECHO OPTIONAL)
  - DYNAMIC MODE CHANGES
  - CONFIGURABLE
    - FEATURES
    - CRT's
    - PORTS
  - ALL ON EACH USART!

544 DRIVER

- MMX BASED
  - ON BOARD EDITING AND USART HANDLING

iRMX 86 RELEASE 5

INTERACTIVE CONFIGURATION UTILITY (ICU)

- UDI BASED (WILL RUN ON SYSTEM III)
- EASY TO USE QUESTION/ANSWER
- CONFIGURATION PROCESS:
  1. LAYOUT APPLICATION
     JOBS, TASKS, ENCHANGES
  2. WRITE, COMPILE, LINK, LOCATE APPLICATION CODE
  3. USE ICU TO GENERATE DESCRIPTION FILE
  4. ICU/GENERATE COMMAND
     CONFIGURATION MODULES
     SUBMIT FILE (ICU86)
  5. PARTITION SUBMIT FILE IF FLOPPY SYSTEM
  6. SUBMIT ICU86
  7. TEST
  8. REDO STEPS 1 - 7 IF NECESSARY
  9. FINAL RECONFIGURE FOR MINIMUM SYSTEM
     SYSTEM CALLS
     DEBUGGER REMOVAL

iRMX 86 RELEASE 5

INVOCATION EXAMPLE

ICU 86 (INPUT FILE NAME TO) OUTPUT FILE NAME

- NO INPUT FILE NAME: SYSTEM DEFAULT DESCRIPTOR FILE USED AS INPUT
- DUPLICATE OUTPUT FILE NAME: OLD FILE COPIED TO "OUTPUT FILE NAME BAK"

iRUG

LIFEBOAT/INTEL COMMITMENTS

LIFEBOAT WILL PROVIDE THE MAJOR FUNDING AND WILL ORGANIZE AND MANAGE A CLERICAL, TECHNICAL AND PUBLICATION STAFF FORTHE USER GROUP. THEY WILL ACTIVELY PROMOTE, ADVERTISE AND DISTRIBUTE USER-GENERATED SOFTWARE TO BE CATALOGED AND INCORPORATED INTO AN iRUG LIBRARY.

INTEL WILL PROVIDE PARTIAL FUNDING, AN86/330 SYSTEM AND TRAINING TO SUPPORT LIFEBOAT'S ENDEAVORS.