ปีการศึกษา 2533

การควบคุม เครื่อง LEAD BOND โดย VAX COMPUTER

โดย

นาย นิมิต จันทร์ด้ง

นาย วิทยา เกิดสันเทียะ

นาย เจริญเกียรติ หุตะนานันทะ

อาจารย์ที่ปรึกษา

อาจารย์ ภากร หุตะสังกาศ

ปริญญานิพนธ์ปีการศึกษา 2533

เรื่อง การควบคุม เครื่อง LEAD BOND โดย VAX COMPUTER

ผู้จัดทำ.....

1. นาย นิมิต จันทร์ด้ง
2. นาย วิทยา เกิดสันเทียะ
3. นาย เจริญเกียรติ หุตะนานันทะ

..... อาจารย์ที่ปรึกษา

(.....)

การควบคุม เครื่อง LEAD BOND โดย VAX COMPUTER

นาย นิมิต จันทร์อ้ง

นาย วิทยา เกิดสันเทียะ

นาย เจริญเกียรติ หุตะนานันทะ

อาจารย์ที่ปรึกษา อาจารย์ ภากร หุตะสังกาศ

ปีการศึกษา 2533

กิตติกรรมประกาศ

โครงการนี้ได้รับความร่วมมือจากหลายฝ่ายเพื่อจะสื่อสารเชื่อมโยง เทคโนโลยีเข้าด้วยกันให้ได้ผลงานออกมาอยู่ในรูป AUTOMATION และสอดคล้องกับเป้าหมายของโครงงาน โดย ระบบทางด้าน HOST COMPUTER VAX SYSTEM ได้รับความร่วมมือจาก คุณ วราวุธ ศุภรัตโนดม ซึ่งให้ความช่วยเหลือในด้าน SYSTEM MANAGER ของ COMPUTER ทางด้าน DATABASE MANAGEMENT ได้รับความช่วยเหลือจาก คุณ มุกดา องคสุวรรณ และทาง ด้าน COMPUTER HARDWARE,STANDARD SECS II PROTOCAL จาก คุณ JUAY CHIP HENG และ คุณ TH ONG

ทางคณะผู้จัดทำขอขอบคุณบุคคลดังกล่าวที่ได้ให้ความรู้ และให้ความแนะนำมาประกอบกับการพัฒนาโครงงานให้สามารถบรรลุเป้าหมาย ไว้ ณ. ที่นี้

ขอแสดงความขอบคุณ

คณะผู้จัดทำ

10 ตุลาคม 2533

คำนำ

ปัจจุบันโรงงานส่วนใหญ่จะมีการใช้เครื่องมือในการผลิต ที่มีการควบคุมด้วยระบบ COMPUTER ซึ่งอาศัยคนป้อนข้อมูลต่างๆ เข้าไปในเครื่อง COMPUTER นั้น และข้อมูลเหล่านั้นจะถูกเก็บไว้ใน หน่วยความจำชั่วคราว (RAM) หรือ หน่วยความจำ ถาวร (DISKETTE, HARD DISK, TAPE) ซึ่งอยู่ในเครื่องที่ใช้งานนี้เท่านั้น ไม่ได้ เก็บไว้ในศูนย์กลางหน่วยความจำ (CENTER MEMORY)

หลักการเช่นนี้ไม่สะดวกในการที่จะใช้งานข้อมูลที่มีอยู่ในแต่ละแผนก (OPERATION) และแผนกอื่นๆ เพราะเมื่อต้องการจะใช้งานข้อมูลเหล่านั้นต้องไปที่เครื่อง(EQUIPMENT) และอ่านข้อมูลที่เครื่องนั้น หรือ นำ DISKETTE ออกมาอ่านที่เครื่องอื่น

อีกประการหนึ่งในการที่เราจะทำเป็นระบบ AUTOMATION นั้นเราจะต้องมี CENTER MEMORY เพราะบางโรงงานจะมีเครื่องเป็นจำนวนมาก ไม่สะดวกในการใช้งานข้อมูลต่าง ๆ หรือประมวลผลหรือแก้ไขทันทีทันใด เพื่อไม่ให้เกิดปัญหาข้อมูลหลัก

โครงงานนี้จัดทำเพื่อรับมูลต่าง ๆ ให้มาอยู่ใน CENTER เดียวกัน เพื่อการ ANALYSIS ปัญหาต่าง ๆ ที่เกิดขึ้น และทำการ INTERFACE HOST COMPUTER กับ EQUIPMENT ต่าง ๆ ทางคณะจัดทำหวังว่าคงเป็นประโยชน์สำหรับผู้ที่สนใจ

คณะผู้จัดทำ

10 ตุลาคม 2533

ABSTRACT

This project explained how to use the computer to control the wire bonding machine(to connect between the chip ic and lead frame). In processing the all data in the wire bonding machine that keep in center memory of host computer, when we need some data for analysis we can get its from this center memory.

It is the DECNET communication system which transfer to VART and send to the wire bonding machine under the SESC PROTOCAL STANDARD, the protocol can control machine by indentify the equipment id for every machine, for the transfer message can specify on form Text, Integer, Floting Point, and Undsigned by Header told the data lenght type and Terminator for support the long data, at the end data it have the chech sum code.

During process it check the processing system by itself for check status the online of inteface between Host and equipment, every few minute Host will can the data at equipment to update at screen monitor(terminal) which it is runing production time, during run process it is real time to up load and down load the data from equipment to Host and Host to equipment by automatically, this point it can reduced the human error.

In software system, the server is assembly language of CPU NS32000, working in multitasking system to communicate between VART and ENTERNET. Host used the software ADA multitasking which it are a multitasking system same as the server, we need

บทคัดย่อ

โครงงานนี้ แสดงถึงการใช้คอมพิวเตอร์ควบคุมเครื่อง WIRE BONDING (เครื่องต่อขาจาก CHIP IC ด้วยลวดทองคำมายัง LEAD FRAME ซึ่งเป็นทองแดง) การควบคุมใช้หลักการที่ว่า ข้อมูลการผลิตและ PROGRAM การผลิตจะนำมาเก็บข้อมูลไว้ที่ DATA CENTER ของ HOST คอมพิวเตอร์ ระบบการสื่อสารที่ใช้ในโครงงานนี้เป็นระบบของ DECNET และผ่านสู่ VART ไปยังเครื่องต่อขา ID ภายใต้ PROTOCOL SECS STANDARD ใน PROTOCOL ตัวนี้จะมีคุณสามบัติคือสามารถควบคุมจำนวนเครื่อง หรือ EQUIPMENT ได้ 2^15เครื่องจาก HOST เพียงตัวเดียวโดยใน PROTOCOL จะมี EQUIPMENT ID อยู่และข้อมูลที่ต้องการส่งสามารถกำหนดชนิดได้เช่น TEXT, INTEGER, FLOATION OPINT, และ UNDSIGNED ต่าง ๆ โดยจะมี HEADER บอกชนิดของ DATA ที่ส่งมาความยาว และการเชื่อมต่อ (TERMINATOR) เพื่อสนับสนุนกรณีมีข้อมูลยาว ๆ เมื่อหมดข้อมูล DAT ในส่วนท้ายก็จะมี CHECK SUM การควบคุมจะมีการรายงานตัวอยู่ตลอดเวลา เพื่อเป็นการ CHECK STATUS การ ONLINE ติดต่อระหว่าง EQUIPMENT กับ HOST การควบคุมของโครงงานนี้ จะประกอบไปด้วย การเรียกการรายงานการทำงานของ EQUIPMENT มา DISPLAY แบบ ONLINE บนคอมพิวเตอร์ TERMINAL การเรียกค่า PARAMETER ของ EQUIMENT ที่ใช้งานในการทำงานผลิตอยู่ ณ ขณะนั้น การเก็บ PROGRAM การทำงานของ EQUIPMENT มาไว้ที่ DATA CENTER (UP LOAD) และการส่งข้อมูลที่ได้เก็บไว้ที่ DATA CENTER ไปให้กับ EQUIPMENT (DOWN LOAD) ทั้งนี้รวมทั้งการเก็บ ข้อมูลเวลาการผลิตในโรงงานและความถูกต้องเที่ยงตรงสูงเนื่องจากเป็นการรับข้อมูลจากเครื่องผลิตโดยตรงโดยผ่านคน จึงสามารถช่วยแก้ไขปัญหา HUMAN ERROR ระบบ SOFTWARE ที่ใช้จะแบ่งเป็นที่ตัว SERVER จะเป็น ASSEMBLY ของ CPU NS32000 แบบ MULTITASKING เพื่อทำการแปลงการติดต่อจาก VART ไปยัง ENTERNET และทางด้าน HOST จะใช้ SOFTWARE ADA MULTITASKING เช่นเดียวกันทั้งนี้เพื่อรองรับการควบคุม EQUIMENTจำนวนมาก.

แนวทางพัฒนา

จากผลการทำวิจัยทางคณะผู้จัดทำมีความเห็นว่า ระบบ PROTOCOL ที่ใช้ใน PROJECT นี้ สามารถนำไปพัฒนาใช้ได้กับระบบ CONTROL FREE RUNING อื่นได้ โดยไม่ต้องขึ้นอยู่กับด้านของ HARDWARE ซึ่งอาจจะใช้ระบบ INFRAFEAD , RADIO FGEQUENCY , หรือระบบสื่อสารที่ใช้สัญญานอย่างเช่น RS232, IEE488 เป็นต้นข้อดีของ SEC RPOTOCOL ที่ใช้ใน PROJECT นี้คือ สามารถส่งข้อมูลได้ทั้งหมด เช่น TEXT, INTERQER ขนาด 1,2,4,8 BYTE FLOTING POINT ขนาด 1,2,4,8, BYTE และ UNSINGNED ขนาดต่าง ๆ ในตัว PROTOCOL ออกแบบรองรับการสื่อสารจาก HOST ไปถึงEQIUQMENTได้ทั้งหมด 2^15 EQUIPMENTซึ่งเป็นจำนวนมากมายที่สามารถรับคำสั่งจาก HOST COMPUTER 1 ตัว และทุกตัวของ EQUIPMENT ก็สามารถรายงานตัวกับ HOST ได้ตลอดเวลา โดยระบบ QUEUE

ในด้านของข้อดีนี้สามารถดัดแปลงเช่น ต้องการควบคุมแขนกล จำนวน 10 ตัว โดยทุกตัวติดต่อผ่าน RADIO FREQUENCY ภายใน PROTOCOL SECSจะสามารถควบคุมการสั่งงานต่าง ๆ ได้จาก HOST ตัวอย่างเช่น HOST ต้องการสั่งให้แขนกลตัวที่ 8 จับชิ้นงานที่กำหนดไว้ใน PROGRAM จาก HOST การทำงานจะเริ่มโดย HOST ทำการ BUILD PROTOCOL ประกอบไปด้วย EQUIPMENT NUMBER&FLAG ในการที่จะบอกกับ EQUIPMENT ว่าต้องการ REPLY ข้อมูลกลับมาจาก EQUIPMENT หรือไม่ข้อมูลหมายเลขของ FUNCTION ซึ่งจุดนี้จะเป็นการกำหนดกันไว้ก่อนระหว่าง EQUIPMENT กับ HOST ตามด้วยความยาวของข้อมูลชนิดและเนื้อหาของข้อมูลตามสุดท้ายด้วย CHECK SUM ซึ่งในครั้งแรกทุก EQUIPMENT จะรับข้อมูลนี้ได้ทั้งหมดทุกตัวแต่ต่อมาถ้า NEMBER ของ EQUIPMENT นั้นไม่ตรงกับ EQUIPMENT ID ของตัวเองก็จะไม่สนใจข้อมูลนั้น ในแต่ละครั้งของการติดต่อภายใต้ SEC PROTOCOL สามารถส่งข้อมูลได้สูงสุด 152 KBYTE ซึ่งพอเพียงสำหรับ ระบบควบคุมแบบ FREERUNNING และเป็นการก้าวเข้าสู่ระะ CIM อย่างแท้จริง และยังเป็นการไม่จำเป็นต้องระบุชนิดของ COMPUTER เพื่อนำการควบคุมเนื่องจาก การควบคุมอยู่ที่มาตรฐาน RPOTOCOL

สารบัญ

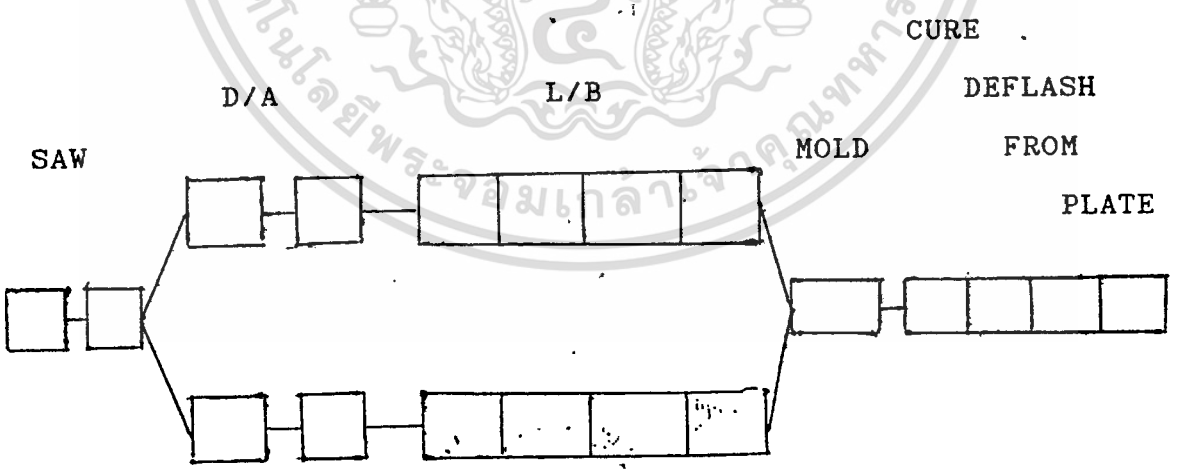
สารบัญ

บทที่ 1

1.1 การทำงานของโรงงานผลิต IC

โรงงานผลิต IC จะประกอบไปด้วย 2 PLANT ใหญ่ๆคือ PLANT ที่ทำหน้าที่เกี่ยวกับการผลิตแผ่น WAFER ซึ่งบรรจุตัว IC เป็นแผ่น SILICON ที่ถูกตัดออกมาเป็นแผ่นบางๆและผ่านการ DOPE สารต่างๆ จากนั้นจะถูกส่งมายัง ASSEMBLY PLANT ทำการตัด WAFER ออกเป็นแต่ละ UNIT จะนำมาติดเข้ากับ FRAME และมีการเชื่อมขาของตัว UNIT ต่อเข้ากับ FRAME และนำ UNIT ที่ต่อเสร็จมาเข้า แผนก MOLD เพื่อปิดตัว IC และส่งไป TEST ก็จะได้เป็น UNIT IC ออกในส่วนของ MOLD นี้อาจจะมีหลายแบบ ซึ่งบางแบบก็ใช้ตัวถังเหล็กแล้วแต่ชนิด

ในส่วนของ โครงการนี้จะมีส่วนเกี่ยวข้องกับ การต่อขางวดทองระหว่างตัว IC UNIT มาที่ FRAME โดยใช้เครื่องชื่อ LEAD BOND ของบริษัท KNS MODLE 1482 ขั้นตอนของ ASSEMBLY PLANT แสดงดัง BLOCK DIAGRAM ต่อไปนี้



*D/A=DIE ATTACH

*L/B=LEAD BOND

รูปที่ 1 LINE การผลิตตัว IC

1.2 HOST NETWORK

NETWORK ของ PROJECT KNS INTERFAEC ประกอบไปด้วย HOST COMPUTER คือ VAX CLUSTER SYSTEM เชื่อมโยงการสื่อสารไประบบ ENTHERNET BUS โดยผ่าน H400 เป็นตัว CLAMPING จาก ENTHERNET BUS ก็จะทำหน้าที่แปลงสัญญาณจากระบบ ENTHERNET เป็นระบบ RS423

ระบบ RS423 มีมาตรฐาน คือ จะมีสัญญาณอยู่ 6 สัญญาณ คือ

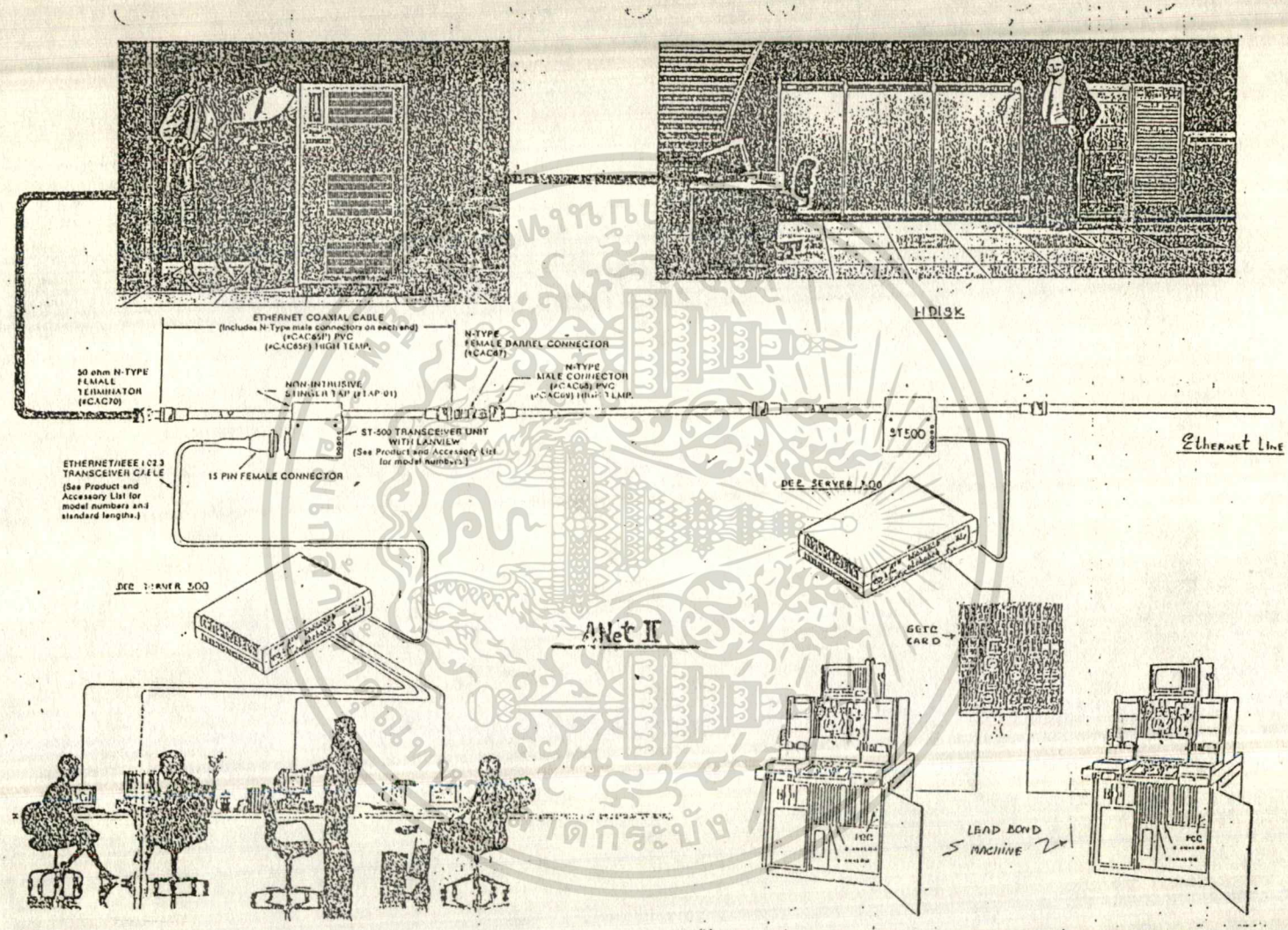
- DTR DATATERMINAL READY
- TRANSMIT DATA
- TRANSMIT COMMON
- RECIVE COMMON
- RECIVE DATA
- DSR DATA SET READY

ซึ่งจากระบบ RS423 นี้สามารถ CONVERSE ไปเป็น RS 232 โดยผ่าน CONVERSE CONNETER ก็จะได้เป็น RS 232 ไปให้กับ EQUIPMENT ซึ่งในที่นี้ก็คือ เครื่อง AUTO MATIC BONDER KNS 1482 เพื่อให้เกิดการสื่อสารกันขึ้นภายใต้ SESC PROTOCAL ทำให้ HOST COMPUTER กับ EQIPMENT สามารถ STATUS กันได้

ทางด้านของ TERMINAL MONITER จะสามารถ CHECK STATUS ของเครื่องได้ว่ากำลัง RUN อยู่หรือ STOP อยู่ในการ CHECK ผ่านทาง PCC CARD ภายใต้ SESC PROTOCAL.

1.3 HOST COMPUTER SYSTEM

HOST COMPUTER เป็นระบบ VAX SYSTEM ซึ่งจัด SYSTEM อยู่ในระบบ


HDISK
ETHERNET COAXIAL CABLE
(Includes N-Type male connectors on each end)
(#CAC65P) PVC
(#CAC65F) HIGH TEMP.
N-TYPE
FEMALE BARREL CONNECTOR
(#CAC07)
50 ohm N-TYPE
FEMALE
TERMINATOR
(#CAC70)
NON-INTRUSIVE
STINGER TAP (#TAP 01)
N-TYPE
MALE CONNECTOR
(#CAC05) PVC
(#CAC09) HIGH TEMP.
ST-500 TRANSCEIVER UNIT
WITH LANVIEW
(See Product and Accessory List
for model numbers.)
ST500
Ethernet Line
ETHERNET/IEEE 802.3
TRANSCEIVER CABLE
(See Product and
Accessory List for
model numbers and
standard lengths.)
15 PIN FEMALE CONNECTOR
DEC SERVER 700
DEC SERVER 300
ANet II
GGTC
CARD
LEAD BOND
MACHINE

3300, UVAX 2000 ในระบบนี้ CPU ทุกตัวสามารถทำงานแทนกันได้และมีศูนย์ข้อมูลแบบ DATA CENTER สามารถ ACCESS ข้อมูลได้สองทิศทางโดยด้านการ CONTROL จาก HSC CONTROLLER เข้าสู่ STORAGE ARRAY และ TAPE DRIVE ทางด้าน การติดต่อศูนย์ข้อมูลจะมีอุปกรณ์ที่ทำหน้าที่ MULTIPLEX อีก UNIT หนึ่งคือ STAR COUPLER ซึ่งจะเป็น UNIT ที่จัดการแบ่งการ CONTROL ให้กับ HSC

การสื่อสารของ CLUSTER SYSTEM จะสื่อสารกับ SYSTEM ทั้งหมดด้วยระบบ EHTERNET โดยการ CLAMP สัญญาณด้วย H4005

ระบบ TERMINAL DISPLAY จะสื่อสารโดยผ่านตัว DECSERVER UNIT ซึ่ง DEC SERUNIT จะเป็นตัวจัดการเกี่ยวกับ การควบคุมระหว่าง TERMINAL กับ COMPUTER SYSTEM ที่ตัว DECSERVER นี้ สื่อสารกับ TERMINAL ด้วยระบบ RS423 ด้วย BONDLATE 19200 มาควบคุมเกี่ยวกับรับข้อมูลและแสดงผลทาง CRT

ระบบ HOST COMPUTER ติดต่อกับ EQUITMET ภายนอกโดยผ่าน RS423 การติดต่อระบบนี้คือจาก CLUSTER COMPUTER จะติอต่อผ่าน H400 เข้าระบบ ENTHERNET และจากนั้น จะผ่าน ENTHERNET เข้า H400 อีกครั้งหนึ่งต่อมาที่ DECSERVER และ OUT PUT จาก DECSERVER จะได้ ระบบสื่อสาร RS423 อีกที่หนึ่งซึ่งสามารถต่อเชื่อมเข้ากับ RS232 ของ EQUITMENT อื่นๆ ได้ซึ่งใน PROJECT นี้จะใช้มาตรฐาน PROJECT SECS STANDARD

1.4 K & S MACHINE

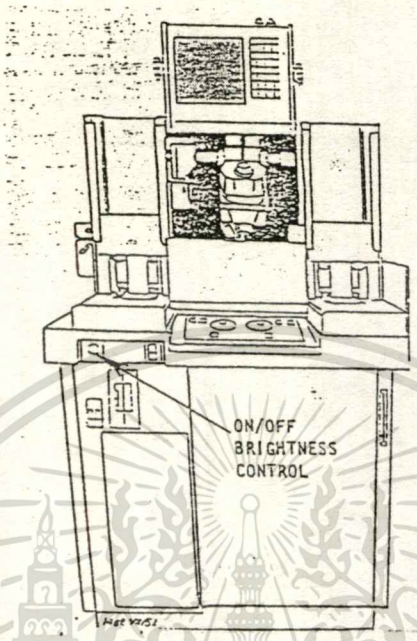
KNS 1482 คือ MODEL ของเครื่อง AUTO LEAD BOND สำหรับทำงานเชื่อมขาของ IC ไปเชื่อมกับ PAD บนตัว DIE ของ IC KNS 1482 นี้จะมีระบบค้นหาภาพโดยอัติโนมัติเพื่อค้นหาแหน่งการต่อขา ในตอนแรก OPERATOR จะเป็นผู้ TEACH PROGRAM ใช้กับเครื่องก่อนและ SET PROGRAM ไว้ใน MEMORY ของเครื่องจากนั้นถ้ามีการสั่ง RUN เครื่องก็จะนำข้อมูลมาค้นหาภาพและการ BOND กับตัว IC ซึ่ง PROGRAM ส่วนนี้จะมีการนำไปเก็บ DATA CENTER ที่ CLUSTER SYSTEM และเป็นจุดประสงค์ของโครงงานนี้

ให้เกิดเป็นลูก BOND ที่ลวดทอง จากนั้ใช้แท่ง CARAMIC มาขยี้ลงบน PAD ของตัว DIE ด้วยการสั่นสะเทือนของ ULTRASONIC เพื่อให้ลวดทองเกาะติดกับ PAD ของตัว DIE เมื่อต่อลวดทองมาที่ตัว DIE ได้ก็จะมีการนำมาต่อเข้ามาที่ LEAD ซึ่งเป็น FRAME ทองแดง โดยการใช้ FORCE กระแทรกเข้ากับ LEAD

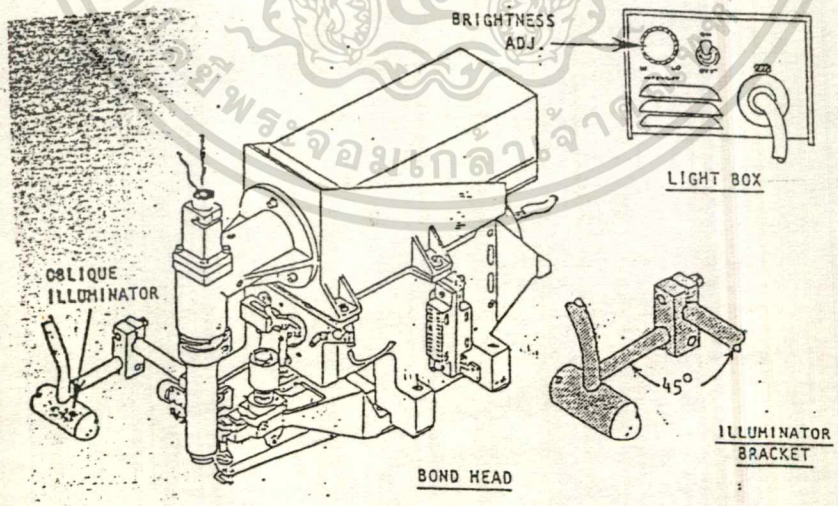
การเก็บ PROGRAM ของเครื่อง KNS 1482 จะแบ่งการเก็บออกเป็น 3 อย่าง คือ ข้อมูลภาพเป็นการจำตำแหน่งภาพ จุดนี้ได้มาจาก OPERATOR จะมีการให้เครื่องจำภาพจุดเริ่มต้นก่อน จากนั้นจะมีข้อมูลของตำแหน่ง X,Y,Z, ซึ่งเครื่องจะรับมาจาก ZEBRA LINE ที่ติดอยู่กับ X,Y,Z, SERVO MOTER และข้อมูลชุดสุดท้ายก็คือข้อมูลของ PARAMETER ต่างๆ เช่น POWER ของ TRANSDUSOR,FORCE,VELOCD หรือความเร็วในการเคลื่อนที่ก่อนจะมีการขยี้ด้วย ULTRASONIC ใน 3 ส่วนนี้จะมี FORMAT การเก็บเป็นรูปแบบของ KNS 1482 แต่งจะสามารถ UP/DOWN LOAD ได้โดยผ่านมาตรฐาน SESC PROTOCAL โดยจะแบ่งเป็น FUNCTION ออกไป

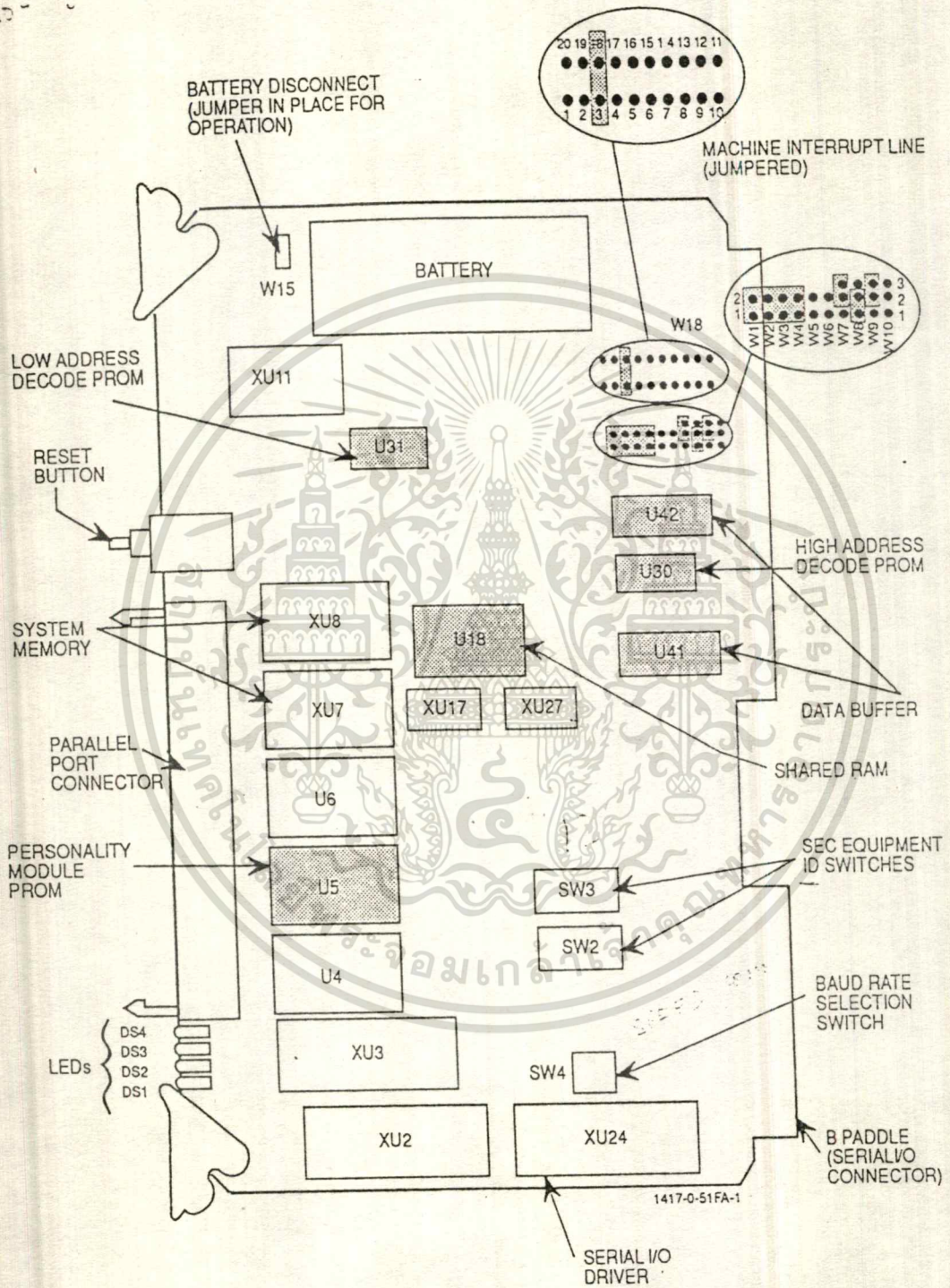
1.5 K & S INTERFACE CARD [PCC]

PCC CARD คือ CARD HARDWARE ที่ทำหน้าที่เกี่ยวกับการสื่อสารทาง RS232 ภายใต้มาตรฐาน SESC PROTOCAL ภายใน CARD นี้จะมีการ SET เกี่ยวกับ DEVICE ID และ BOADLATE ของการสื่อสาร RA232 ซึ่งจะนำไปเชื่อมโยงกับ DECSERVER โดยผ่าน RS423 BUS ออกไป



รูปที่ 3 เครื่อง LEAD BONDING K & S




BATTERY DISCONNECT (JUMPER IN PLACE FOR OPERATION)
MACHINE INTERRUPT LINE (JUMPERED)
W15
BATTERY
W18
LOW ADDRESS DECODE PROM
XU11
U31
RESET BUTTON
U42
HIGH ADDRESS DECODE PROM
U30
SYSTEM MEMORY
XU8
U18
U41
XU7
XU17
XU27
DATA BUFFER
PARALLEL PORT CONNECTOR
SHARED RAM
U6
PERSONALITY MODULE PROM
U5
SEC EQUIPMENT ID SWITCHES
SW3
SW2
U4
BAUD RATE SELECTION SWITCH
LEDs
DS4
DS3
DS2
DS1
XU3
SW4
XU2
XU24
B PADDLE (SERIAL I/O CONNECTOR)
1417-0-51FA-1
SERIAL I/O DRIVER

1.6 K & S SYSTEM NETWORK

K & S SYSTEM NETWORK ประกอบด้วยเครื่อง K&S GOLD WIRE BONDER MODEL 1482 เชื่อมโยงมายัง HOST COMPUTER โดยผ่านระบบ ENTERNET และ RS232 ภายใต้ PROTOCAL SEC II ซึ่งเป็น STANDARD COMMUNICATION ระหว่าง เครื่อง WIRE BONDER กับ HOST COMPUTER

ภายในเครื่อง K&S BONDER EQUIPMENT ประกอบด้วย INTERFACE CARD ชื่อ PCC CARD ทำหน้าที่ CONVERSE สัญญาณจากเครื่อง K&S WIRE BONDER ให้อยู่ภายใต้ SEC PROTOCAL

สัญญาณที่ EQUIPMENT ส่งมายัง HOST COMPUTER ภายใต้ SEC PROTOCAL จะประกอบไปด้วย ABORT TRANSACTION , ARE YOU THERE REQUEST, ON LINE DATA, ALARM REORT SEND ฯลฯ ซึ่งสัญญาณพวกนี้ จะ DESING FORMAT สัญญาณภายใต้ SEC PROTOCAL การติดต่อระหว่าง EQUIPMENT กับ HOST COMPUTER เริ่มจาก HOST COMPUTER จะส่งสัญญาณ ARE YOU THER REQUEST ผ่านสาย RS232 ไปที่ PCC CARD ของเครื่อง K&S WIRE BONDER และตัว PCC CARD ก็จะสร้างสัญญาณตอบกลับ เพื่อให้ HOST COMPUTER รู้ STATUS ว่าขณะนี้ EQUIPMENT ON LINE ALREADY และจะทำให้ EQUIPMENT สามารถ REQUEST LOAD PROGRAM จาก HOST COMPUTER ได้และ HOST COMPUTER สามารถ CHECK STATUS ต่างๆ ของ EQUIPMENT ได้

บทที่ 2

2.1 GEIC SERVER COMPUTER

GEIC-32 คือ GENERIC EQUIPMENT INTERFACE CONTROLLER ในที่นี้ออกแบบไว้สำหรับใช้ CPU แบบ 32 BIT ของ NATIONAL SEMICONDUCTOR SERIES ซึ่งเป็น CPU ที่มีประสิทธิภาพมากในการใช้งานสามารถมีช่วงการทำงานที่กว้างมาก สำหรับการใช้เป็น APPLICATION เกี่ยวกับการ INTERFACE EQUIPMENT กับ HOST COMPUTER และนำไปควบคุม EQUIPMENT ทั้งหมด นอกจากนี้ยังออกแบบในสามารถง่ายต่อการควบคุมแบบ MULTIPLE MACHINE INTERFACE

ในตัวนี้สามารถรองรับ OPTIONALLY INTERFACE คือ ต่อ SERIAL ได้ 8 CHANNEL และมี PROGRAM DEBUGGING FACILITY และจาก BOARD สามารถเชื่อมต่อกับ VAX HOST COMPUTER SYSTEM ได้

2.2 ส่วนประกอบของ GEIC

ใน GEIC-32 ประกอบไปด้วย

- CPU 10 MHZ NS32016-10
- NS32201 ICU TIMING CONTROL UNIT
- MULTI-PROCESSOR BUS INTERFACE
- 512 KBYTES STATIC RAM ON BOARD สามารถต่อ RAM ได้ถึง 8 MBYTE
- UPTO 1 MEYTE PROM ON BOARD
- 4 CNARNNAL RS232 SERIAL COMMUNICATION
- 12 LINE INPUT และ 8 LINE สำหรับ INTERRUPT
- 14 LINE OUTPUT
- 1 PARALLEL PORT FOR INTERFACE VAX

- MICROPROCEESSOR REAL TIME CLOCK

ที่ตัว GEIC นี้สามารถทำงานได้ 2 แบบคือ STAND ALONE และ HOST ASSISTED เพื่อสะดวกต่อการนำไปใช้งานควบคุม (CONTROL ต่างๆ)

-การ RUN MODE STAND ALONE

ใน MODE STAND ALONE GEIC-32 VOARD จะ RUN APPLICATION PROGRAM PROM ภายในตัว BOARD โดยไม่ต้องมาอาศัยจาก HOST COMPUTER SYSTEM

-การ RUN MODE ASSISTED

ใน HOST ASSISTED MODE GEIC-32 จะใช้หารเชื่อมโยงข้อมูลโดย SERIES 32000 GENIC NATIVE และ CROSS-SUPPORT (GNX) ซึ่งเช่น PROGRAM ช่วยในการติดต่อใน HOST COMPUTER VAX สามารถติดต่อกับ GEIC-32 ภายใน NATIVE PROGRAM นี้จะมี FUNCTIONS ที่จะ XECVTION และ DEBUG กับ GEIC MONITER PROGRAM ที่ใช้อยู่บน GEIC BOARD และสามารถ UNLOAD PROGRAM โดยผ่านการควบคุมจาก HOST COMPUTER PHYSICAL DESCRIPTION

2.3 DEVENLOPMENT TOOLS

ใน SERIES 32000 MILROPROCESSOR FAMILY มี SOFWARE TOOL SUPPORT อยู่ 4 ชุด เพื่อควบคุมการออกแบบใช้งาน GEIC

- NASM เป็น ASSEMBLER FOR THE ENY ASSEMBLER LANGUAGE SOURE CODE
- NMELD เป็น SOFTWARE LINK EDITOR สำหรับแก้ไข PROGRAM ระหว่าง CPU กำลัง RUN งานอยู่าโดยจะมีการ RELOCATABLE ADDRESS เพื่อแบ่งช่วงการทำงาน
- DBG16 เป็น SOFTWARE DEBUGGER สำหรับ DOULOAD CODE GNX ใช้กับ GEIC BOARD
- NBURN เป็น SOFTWARE PROM PROGRAMMING UTILITY

ที่ตัว GEIC นี้มี SOFTWARE COMPATIBLE WITH VERSATILE REAL TIME

EXECUTIVE ออกแบบโดย โรงงาน NATIONAL SEMICONDUCTOR สำหรับรองรับ

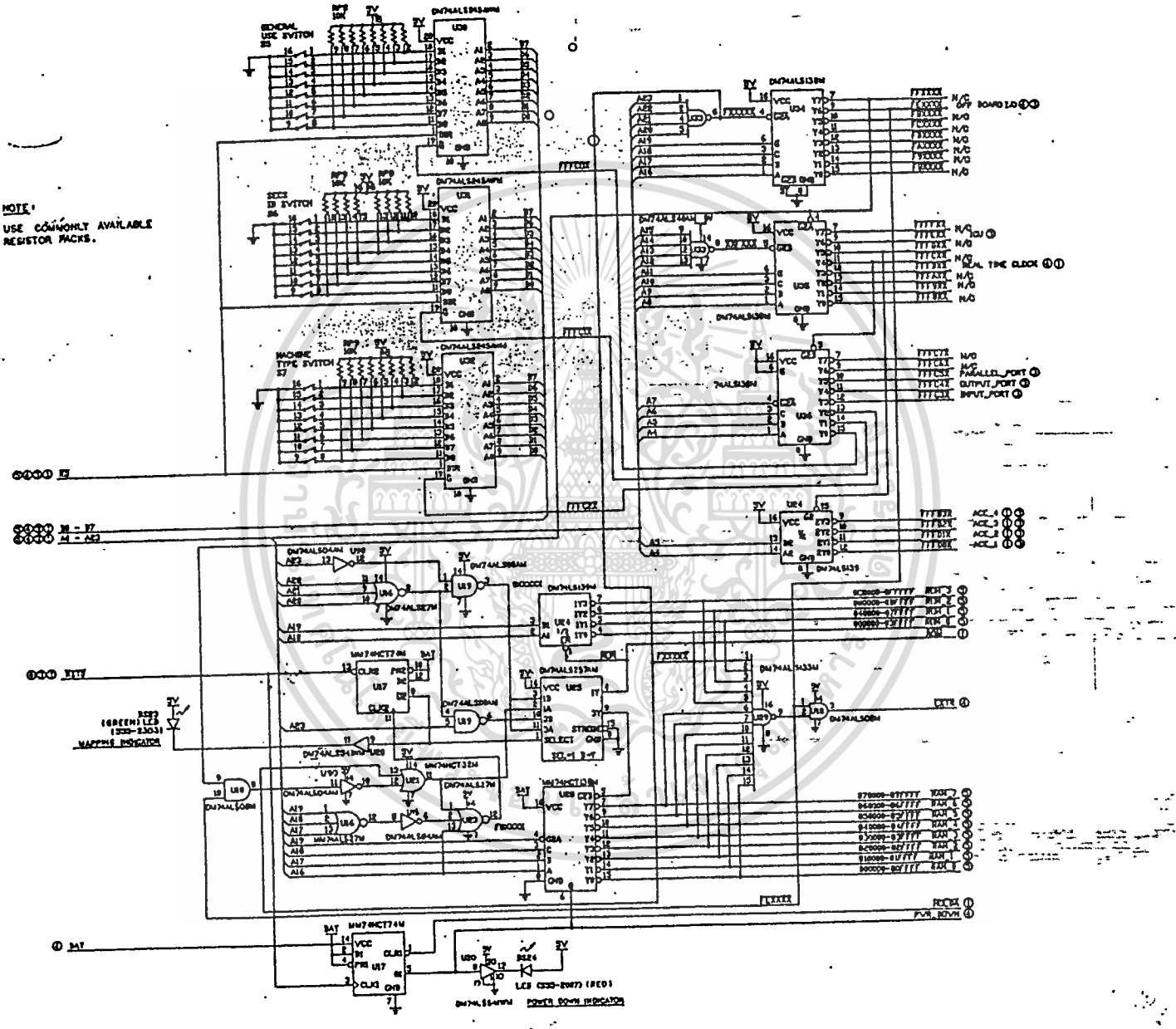
งาน REAL]TIME ในด้าน

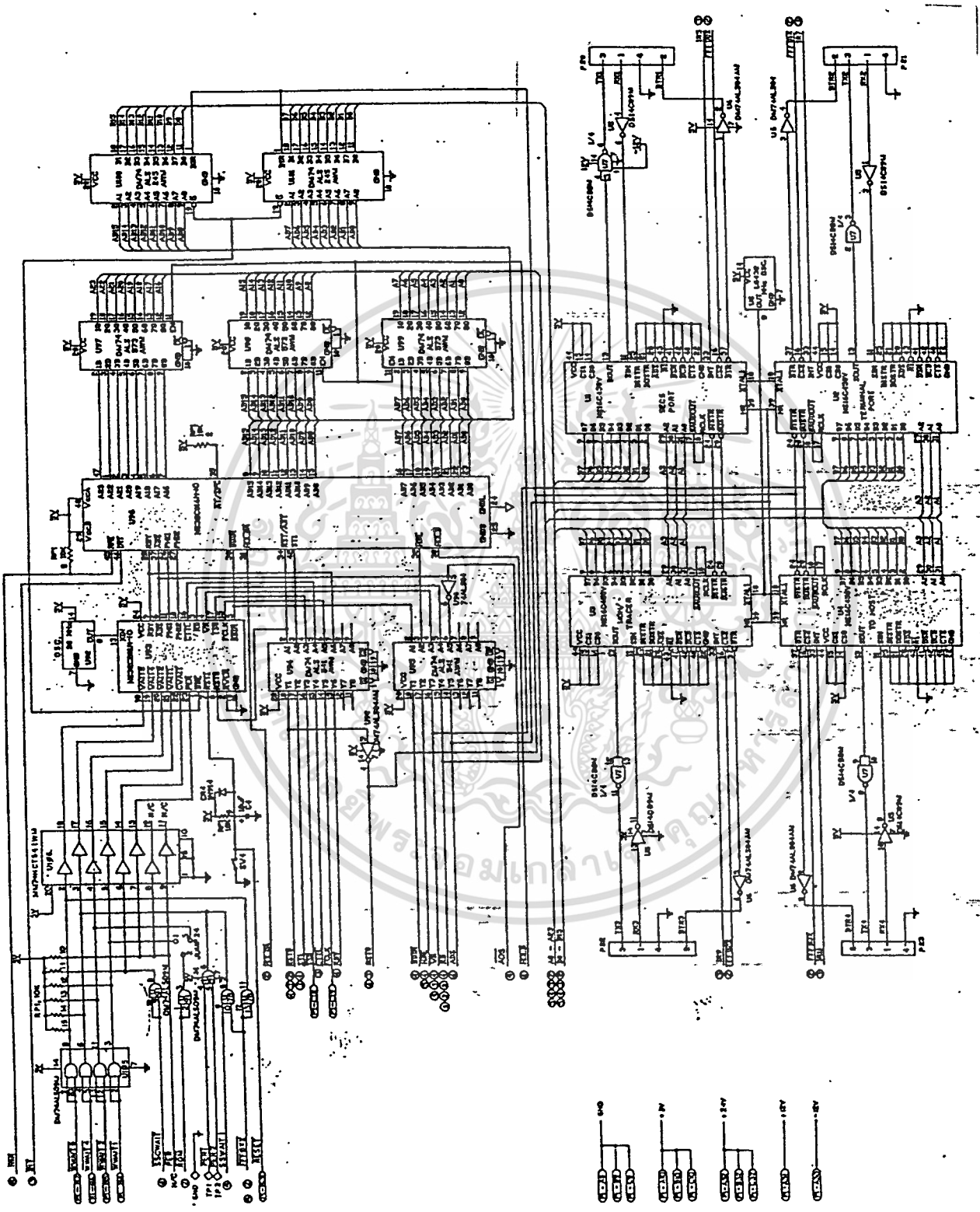
- MULTITASKING SUPPORT
- EVEMT] DROVEM [ROPROTU] BASE SCHEDULING
- INTERTASK COMMUNICATION AND SYNCHRONIZATION
- DYNAMIC MEMORY ALLOCATION
- REAL]TIME CLOCK CONTROL
- CHARACTER I/O SUPPORT
- REAL-TIME RESPONSIVENESS

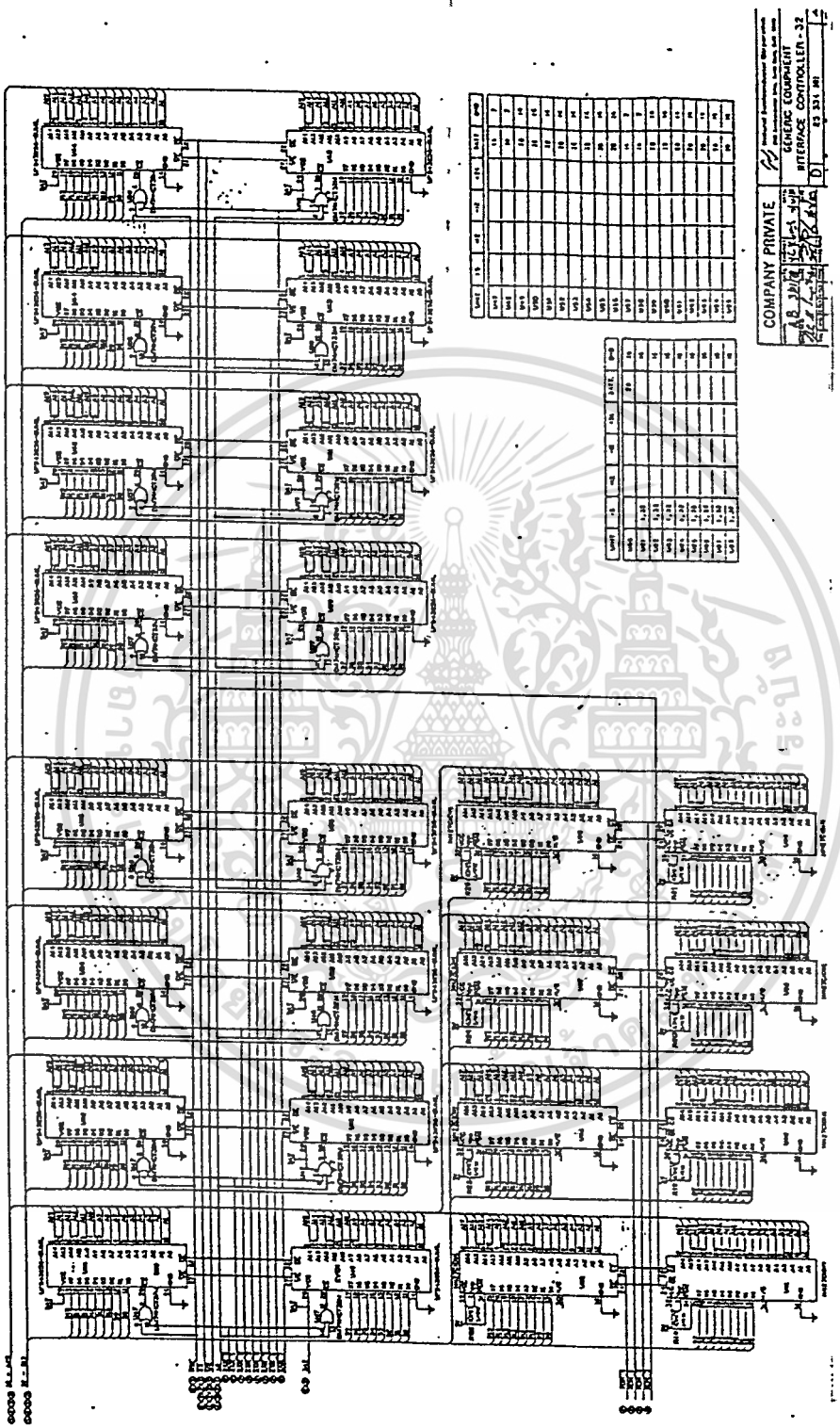
GEIC-32 เป็น COMPUTERCONTROL แบบ MULTI-LAYER ขนาด 5.2X8.9 นิ้ว ประกอบด้วย ONNECTOR INPUT OUTPUT COMMUNICATE

- 96 PIN CONNECTOR ตามมาตรฐาน DIN41612 ซึ่งเชื่อมต่อโดยตรงกับ CPU
- 26 PIN สำหรับ LINKTO VAX SYSTEM PROGRAM I/O
- SWITCH S1 คือ S1F1 SWITCH, S2 SINGLE STEP, S3 NMI NORML NITERUPT SWITCH, S4 คือ RESET
- DIP SWITCH, S5 GENERAL USER, S6 SECS ID, S7 SECSTYPE SELECTOR
- SERIAL I/O CONNECTORS คือ J20-J23
- 20 LED INDICATORS สำหรับ MONITER SYSTEM.

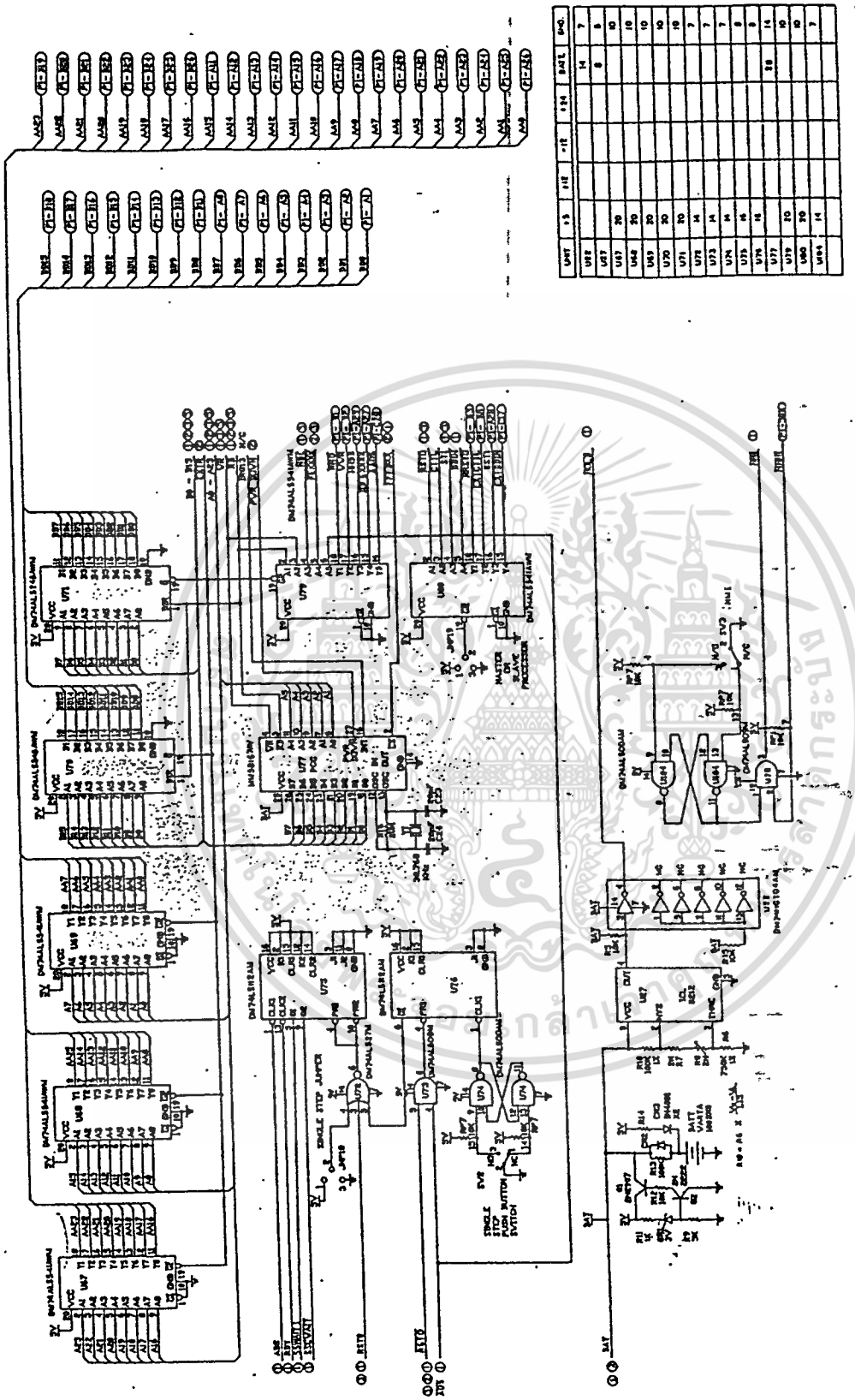
2.4 วงจรภายใน ของ GEIC

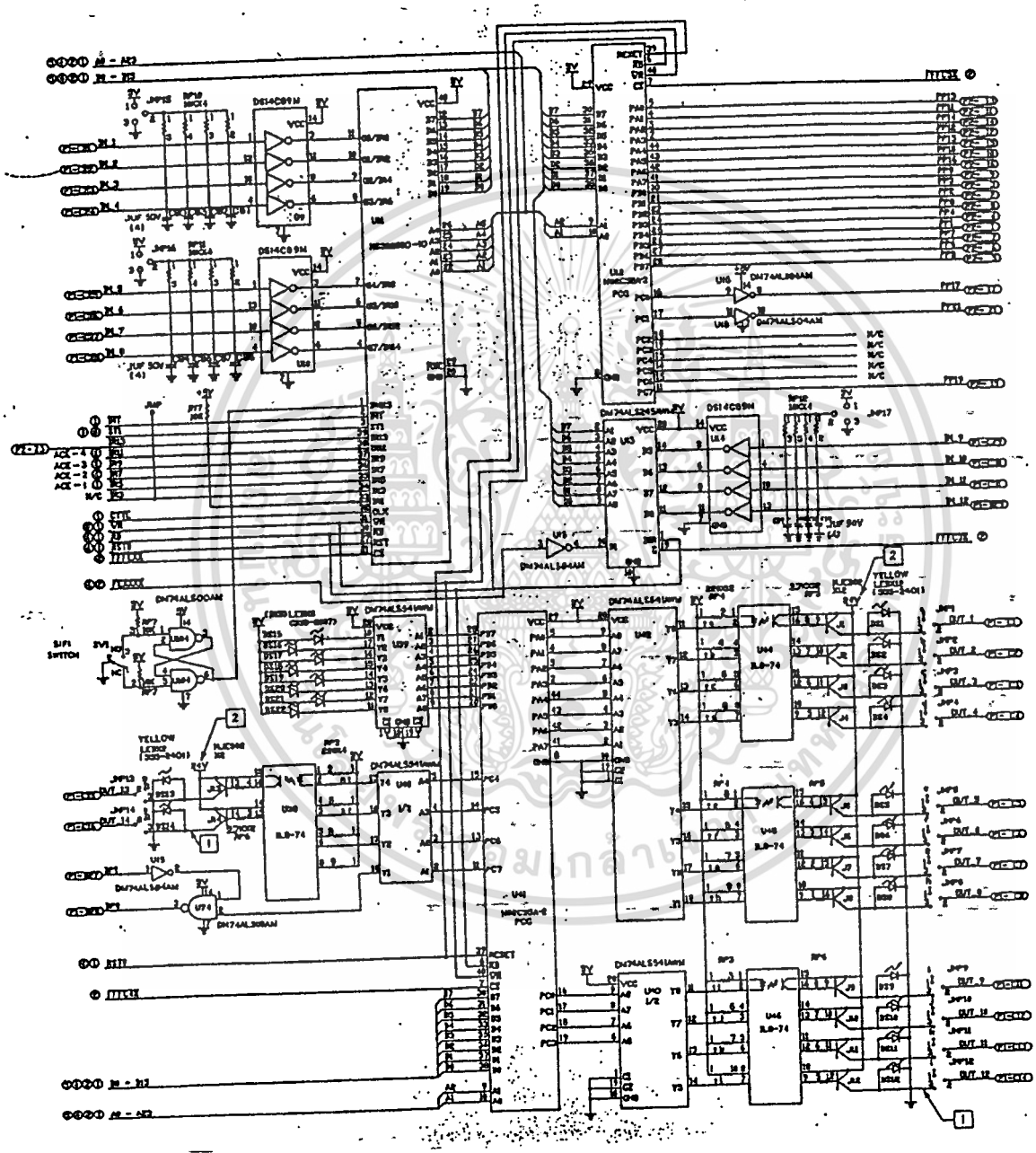

NOTE:
USE COMMONLY AVAILABLE RESISTOR PACKS.
SERIAL USE SWITCH
MACHINE TYPE SWITCH
BOARD I/O
REAL TIME CLOCK
PARALLEL_PORT
OUTPUT_PORT
INPUT_PORT
MAPPING INDICATOR
POWER DOWN INDICATOR
STROBE SELECT





รูปที่ 8 วงจร GEIC (3)





รูปที่ 10 วงจร GEIC (5)

บทที่ 3

3.1 ภาษา ADA

ADA เป็นโปรแกรม LANGUAGE เอนกประสงค์ ซึ่งถูกสร้างขึ้นสำหรับเขียน โปรแกรมที่มีระบบใหญ่มากๆ และโปรแกรม สำหรับ REALTIME SYSTEM ภาษา ADA นี้มีโครงสร้างภาษาแบบ STRONGLY TYPED สามารถนำโปรแกรมอื่นเข้ามาทำงานร่วมกันได้และสามารถเพิ่ม FUNCTION ทางคณิตศาสตร์ชั้นสูงได้ มาตรฐานของภาษา ADA นี้มี SPECIFICED IN AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)/MIL-STD-1815A-1983 AND INTENATIONAL STANDARDS ORGANIZATION (ISO)/8652-1987

สำหรับ ADA ที่ใช้อยู่ในโครงงานนี้เป็นแบบ VAX ADA ซึ่งทางบริษัท DIGITALได้พัฒนาขึ้น โดยพื้นฐานจะมีมาตรฐานของคำสั่งเป็นไปตาม SPEC ของ ADA STANDARDS และมีการเพิ่มเติมเกี่ยวกับ INTERACTS LANGUAGE เช่น RECROD MANAGEMENT,COMDITION HANDING FACILITY

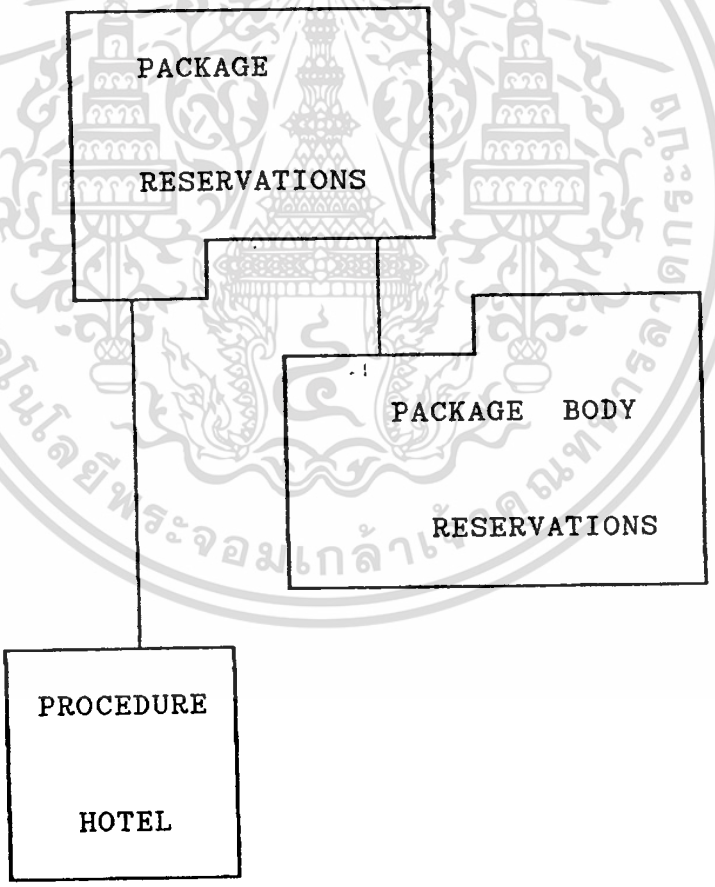
VAX ADA RUN อยู่บน VMS SYSTEM (VAX MANAGEMENT SYSTEM) ซึ่งเป็นมาตรฐานของ LANGUAGE ระบบ VAX ซึ่งคล้ายกับ UNIX SYSTEM หรือ DOS SYSTEM

3.2 การทำงานเกี่ยวกับโปรแกรม ADA

การสร้าง VAX ADA PROGRAM จะมีขั้นตอนดังนี้

- สร้าง WROKING DIRECTORY สำหรับเก็บ ADA SOUREC FILES และมีการสร้าง DIRECTORY มาตรฐาน สำหรับที่จะ EDITING แล DEBUGING
- CREATE A PROGRAM LIBRARY
- DEFINE A CURRENT PROGRAM LIBRARY FOR OPERATIONS SUCH AS COMPILATION,RECOMPILATION, AND SO ON.

- EXECUTE THE PROGRAM.
- DEBUG THE PROGRAM, IF NECESSARY.
- GO BACK TO STEP 5, AND COMPILE THE PROGRAM AGAIN IF DEBUGGING HAS RESULTED IN MODIFICATIONS TO THE SOURCE FILES.



รูปที่ 3 ผังการทำงานของภาษา ADA

3.3 SOFTWARE

K&S NETWORK SYSTEM ประกอบด้วย SOFTWARE

1. STATUS MONITORLING.
2. DATALOGGING.
3. PARAMETERS CONFIGURATION.
4. DEVICE PROGRAMS SUB-MODULES.
5. PASSWORD MAINTENAEC SUB-MODULES.
6. DOWNTIME TRANSACION REPORT.
7. DAILY AND WEEKLY REPORTS.

3.3.1 STATUS MONITORING.

รายงาน STATUS เป็นตัวย่อตัวอักษร 85 MACHINE พร้อมทั้งสามารถ SELECT MACHINE เข้าไปดูรายละเอียด

1.1 DEVICE NAME (MAXIMUM 16 CHARACTERS DISPLAY OVER TWO LINES)

1.2 DEVICE ID (MAXIMUM 6 CHARACTERS)

1.3 BONDER MODEL

1.4 SOFTWARE REVISION

1.5 PREVERNTIVE MAINTENANCE DATE

1.6 CURRENT MODE

1.7 CURRENT STATE

1.8 PARAMETER THAT FALLS OUT OF LIMIT STATUS

3.3.2 DATALOGING

รายงานรายละเอียดเป็นเครื่อง โดยประกอบด้วย

2.1 BONDER NO.

2.3 SOFTWARE REVISION

2.4 DATE OF CURRENT STATE

2.5 TIME OF CURRENT STATE

2.6 CURRENT STATE

2.7 DATE OF CURRENT MODE

2.8 TIME OF CURRENT MODE

2.9 CURRENT MODE

2.10 DATE OF COMMUNICATION DURING SETUP

2.11 TIME OF COMMUNICATION DURING SETUP

2.12 COMMUNICATION LINK FURNIG SETUP

2.13 DATE OF LAST ALARM

2.14 TIME OF LAST ALARM

2.15 LAST ALARM TEXT

2.16 STOP FLAG ("TRUE" INDICATES THAT BONDER STOP DUE TO ONE OF THE BONDER PARAMETERS FALLING OUTSIDE THE LIMIT, OTHERWISE "FALSE")

3.3 การสื่อสาร INPUT/OUTPUT ของ HOST COMPUTER

การส่ง DATA ควบคุม จาก HOST COMPUTER ผ่าน RS423 ไปยัง EQUIPMENT การส่งข้อมูลจะออกจาก PROGRAM ภาษา ADA ซึ่งใช้ในโครงงานนี้โดย PROGRAM จะเรียก SYSTEM FUNCTION ด้วยการ CALL PACKAGE SYSTEM, STARLER ซึ่งเกี่ยวกับการกำหนด ตัวหนด BUFFER และ CONDITION HANDLING สำหรับบอก ERROR ต่างๆและยังมี TASKING-SERVICES ที่จะเป็น RUTINE ของ SYSTEM UNCTION SYSTIM FUNCTION ที่ควบคุมเกี่ยวกับการ OUT DAT คือ QIOW COMMAND FUNCTION PARAMETER ภายในประกอบไปด้วย

1. STATUS คือ CONDITION ที่จะคอยกลับมาว่า ได้ทำการวาง DATA ออกไป

2. IOSE คือ CONDITION VALUE ที่ตอบกลับเช่นกันมีความหมายคือการส่ง DATA ออกไปหรือว่ารับ DATA เข้ามา COMPLETE หรือไม่

3. CHAN คือตำแหน่ง CHANNAL ของ OUT PUT PORT ซึ่งจะอ้าง NODE และ PORT สำหรับ NODE คือ ID ของตัว DECSERVER ซึ่งจะเป็น ENTHERNET ID และ PORT คือตำแหน่ง PORT ที่ต้องการ SET เป็น INPUT OUTPUT PORT การใช้งานจะต้องถูก DEFINE จาก DENET MANAGTEMENT SYSTEM ซึ่งมีชื่อว่า NCP (NET WORK CONTROL PROGRAM) สำหรับกำหนดชนิด (TYPE) ของ PORT นั้น การทำการ SET จะต้องผ่าน PRIVILAGE ของ STEM MANGER เท่านั้น

4. FUNC FUNCTION นี้เช่นค่าตัวแปรของ POUTINE SUPPORT อีกที่หนึ่งซึ่งจะ SPPORT เกี่ยวกับ INPUT, TIME OUT, OUT PUT

5. P1-P6 เช่น PARAMETER ที่จะสนับสนุน RUTINE SUPPORT อีกที่หนึ่งในที่นี้จะใช้งานเพียงแปรเท่านั้นคือ

- P1 BUFFER DATA
- P2 LENGTH
- P3 TIME OUT
- P4 TERMINATOR

การใช้งานเบื้องต้นจะเป็น PROGRAM ดัง ตัวอย่างต่อไปนี้

```
-- การ เรียกใช้ PACKAGE ต่างๆ --
with TEXT_IO;use TEXT_IO;
with STARLET;use STARLET;
with SYSTEM;use SYSTEM;
with CONDITION_HANDLING;use CONDITION_HANDLING;
with RTL;
with MSV_PIC_PKG;
with TASKING_SERVICES;
with ADU; use ADU;
```

```
procedure QIOW is
        DECSERVER    : CHANNEL_TYPE;        -- การกำหนดตัวแปร --
        STATUS       : COND_VALUE_TYPE;
        SENDMSG      : STRING(1..1) := "a";
        SENDMSGLEN   : INTEGER := 1;
        IOSB_VAR     : IO_STATUS_BLOCK_TYPE;
        RECMSG       : CHARACTER := ' ';
        RECMSGLEN    : INTEGER := 1;
        TIMEOUT      : INTEGER := 10;
        TERMINATOR   : INTEGER := 0;

        x            : STRING(1..2) := "  ";
        y            : integer := 0;
        ACK          : constant CHARACTER := ASCII.ACK;
        ENQ          : constant CHARACTER := ASCII.ENQ;
        EOT          : constant CHARACTER := ASCII.EOT;
        NAK          : constant CHARACTER := ASCII.NAK;
        NUL          : constant CHARACTER := ASCII.NUL;
        SOH          : constant CHARACTER := ASCII.SOH;
        MIT          : SYSTEM.UNSIGNED_BYTE;
        BIT8         : SYSTEM.BIT_ARRAY_8;
        STRBIT       : STRING(1..8) := (OTHERS => '0');
        --การสร้าง ฟังชั่น--
        function CHARACTER_TO_UNSIGNED_BYTE is new
             UNCHECKED_CONVERSION ( SOURCE => CHARACTER,
                                    TARGET => SYSTEM.UNSIGNED_BYTE);
        function SH_INT_TO_STR(NUM : in INTEGER)
```

```
LEN    : INTEGER := INTEGER'IMAGE(NUM)'LENGTH;
  BUF    : STRING(1..LEN) := INTEGER'IMAGE(NUM);
  begin
     if NUM < 10 then
        BUF(1..1) := "0";
       return BUF(1..LEN);
     else
       return BUF(2..LEN);
     end if;
  end SH_INT_TO_STR;
 --การกำหนด ช่องทางสื่อสาร ของ HOST COMPUTER--
 procedure PORT_ASSIGN is

  STATUS      : COND_VALUE_TYPE;
  begin
    ASSIGN ( STATUS => STATUS,
    CHAN    => DECSERVER,
    DEVNAM => "LTA3213:");
     if not SUCCESS(STATUS) then
         PUT_LINE("Failed to associate input device");
     end if;
 end PORT_ASSIGN;
 ------------------------------------------------------
 begin
   INIT_ADU;
   PORT_ASSIGN
   PUT_LINE("Working.....................");
```

```
                                    CHAN     => DECSERVER,
--คำสั่ง ส่งข้อมูลออก--             FUNC     => IO_WRITELBLK,
                                    IOSB     => IOSB_VAR,
   P1       => SYSTEM.TO_UNSIGNED_LONGWORD(ENQ'ADDRESS),
   P2       => SYSTEM.UNSIGNED_LONGWORD(1) );
   if not SUCCESS(STATUS) then
    PUT_LINE(".............Sorry SEND Unsuccessfull");
   else
    PUT_LINE(".............OH HO SEND Successfull");
   end if;
   loop

     RECMSG := NUL;
     X    := "  ";
    TASKING_SERVICES.TASK_QIOW( STATUS  => STATUS,
    --คำสั่ง รับข้อมูลเข้า host--    CHAN    => DECSERVER,
                    FUNC     => IO_READLBLK + IO_M_TIMED,
                   IOSB     => IOSB_VAR,
      P1 => SYSTEM.TO_UNSIGNED_LONGWORD(RECMSG'ADDRESS),
     P2  => SYSTEM.UNSIGNED_LONGWORD(RECMSGLEN),
     P3  => SYSTEM.UNSIGNED_LONGWORD(TIMEOUT),
     P4  => SYSTEM.UNSIGNED_LONGWORD(TERMINATOR) );
if not SUCCESS(STATUS) then
 PUT_LINE(".............Sorry RECEIVE Unsuccessfull");
end if;
if integer(IOSB_VAR.STATUS) = STARLET.SS_TIMEOUT then
   PUT_LINE("*** SO SORRY -----> TIMEOUT........."    );
```

```
      end if;
      --การเปรียบเทียบ ascii code--
      if RECMSG = ENQ then
         put_line("This is ENQ");
      elsif RECMSG = EOT then
         PUT_LINE("This is EOT");
      elsif RECMSG = ACK then
         PUT_LINE("This is ACK");
      elsif RECMSG = NAK then
        PUT_LINE("This is NAK");
      end if;
      X   := " "&RECMSG;
     MIT  := CHARACTER_TO_UNSIGNED_BYTE(RECMSG);
     BIT8 := SYSTEM.TO_BIT_ARRAY_8(MIT);
     STRBIT(1..8) := (OTHERS => '0');
     for I in 1..8 loop
         if bit8(i-1) then
            STRBIT(9-I..9-I) := "1";
         end if;
     end loop;          --การแสดงผล--
     PUT_LINE(X&" "&SYSTEM.UNSIGNED_BYTE'image(MIT)&
        " "&STRBIT(1..4) &
        " "&STRBIT(5..8)&"       ("  & INTEGER'IMAGE(Y));
     Y := Y + 1;
     end loop;
end QIOW;
```

บทที่ 4

4.1 STANDARD PROTOCOL SECS.

SEMICONDUCTOR EQUIPMENT COMMUNICATION STANDARD ซึ่งวางระบบโดย SEMI (SEMICONDUTOR EQUIPMENT AND MATERIAL & INSTITUTE INC) และออกแบบ SECS ออกมา 2 ชุดคือ SECS1 และ SECS2 SECS1 นี้ใช้เป็นการสื่อสารมาตรฐานเกี่ยวกับการเชื่อมโยงการควบคุม ข้อมูล (DATA LINK CONTROL) และ SECS2 นี้เป็นมาตรฐานการสื่อสารแบบ MASSAGE FORMAT

PROTOCOL OVERVIEW

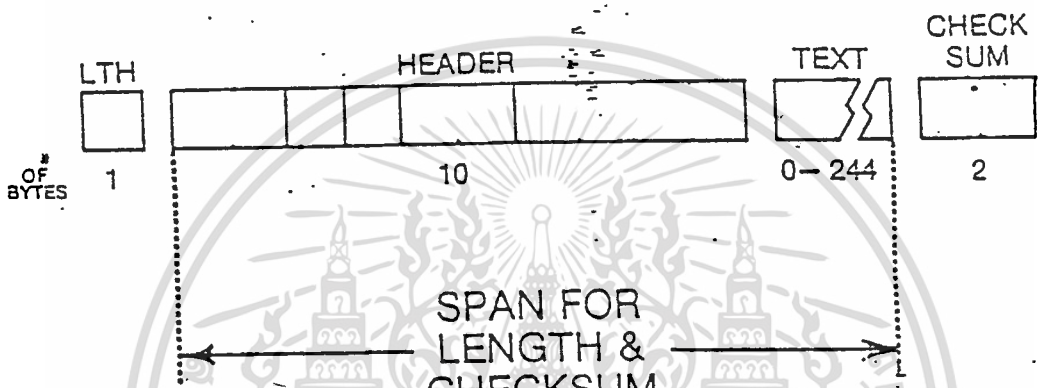
มาตรฐาน SECS กำหนดให้ PROTOCOL ของการสื่อสารระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ ที่ใช้ควบคุมเครื่องที่ใช้งานผลิต IC ในโรงงานผลิต IC (IC MANUFACTURING) ระบบที่ใช้สื่อสารมาตรฐานที่ใช้คือ RS232C STANDARD โดยใช้ขาสัญญาณ

- DTR DATA TERMINAL READY (OUT PUT SIGNAL)
- RTS REQUEST TO SEND (OUT PUT SIGNAL)
- DSR DATA SET READY (INPUT SIGNAL)
- CTS CLEAR TO SEND SEND (INPUT SIGNAL)

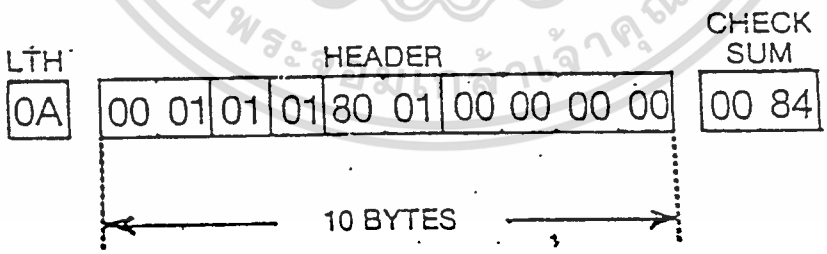
MESSAGE PORMAT

SECS MESSAGE BLOCK ประกอบด้วย ความยาวของ BYTE HEADER (10 BYTES), TEXT (0-244 BYTES) และ CHECK SUM (2 BYTE) สำหรับ LENGTH BYTE หรือความยาวของ BYTE จะเป็นแบบ UNSIGNED BINARY FORMAT และไม่รวมถึง BYTE ของ CHECK SUM "ขนาดต่ำสุดของ BLOCK SIC" จะประกอบไปด้วย HEADER และ TEXT ความยาว BYTE (LENGTH BYTE) เท่ากับ 10 ถ้ารวมทั้งหมดก็จะได้ 13 BYTE คือ LENGTH 1 BYTE HEADER 10 BYTE และ CHECK SUM 2 BYTE ขนาดสูงสุดของ BLOCK SIEB ประกอบด้วย HEADER TEXT

SECS MESSAGE FORMAT



AN ACTUAL MESSAGE

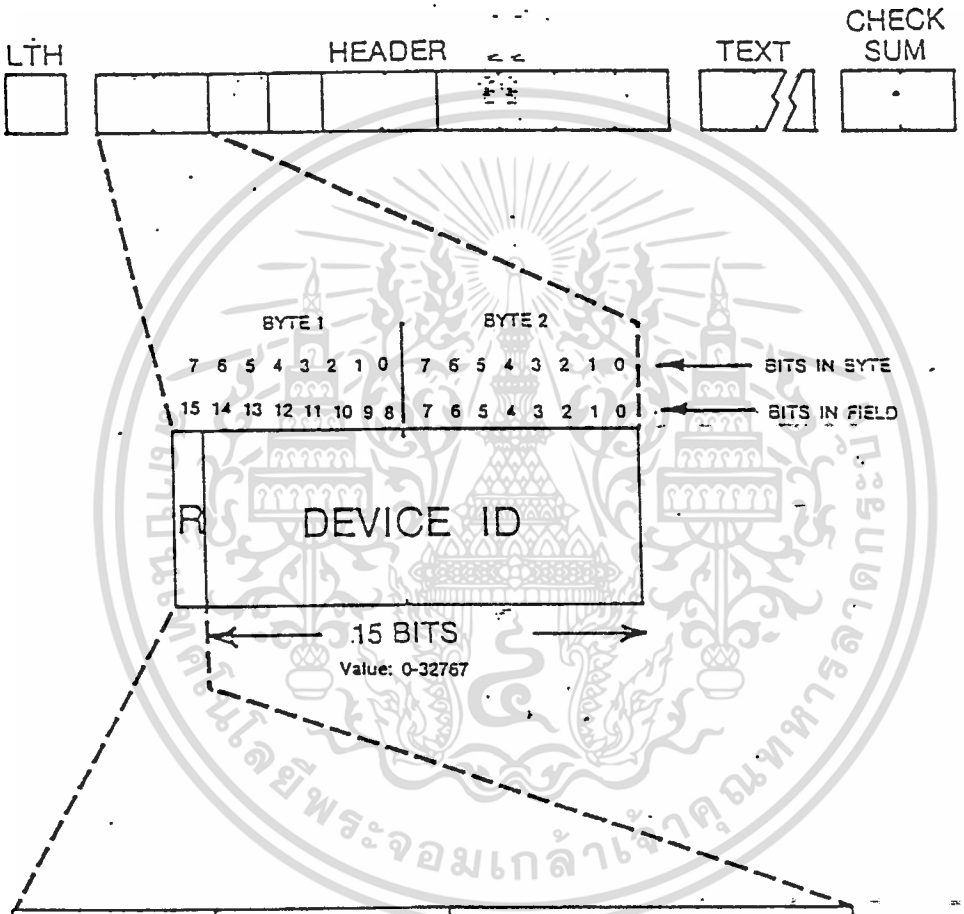


อีก 244 BYTE ถ้ารวมทั้งหมดก็จะเท่ากับ 257 BYTE คือ HEADER 10 LENGTH BYTE 1 TEXT 244 CHECK SUM 2 ทั้งหมดก็จะได้ 257 ในด้านของ CHECK SUM จะเป็นการคำนวณ แบบ 16 BIT ตั้งแต่ HEADER ถึง TEXT โดย CHECK SUM ปกติได้ "0" และค่าใช้งานจริงคือ ค่า SUM ของ UNSIGNED 16 BIT ของ HEADER และ TEXT แต่จะเก็บค่าเฉพาะ 2 BYTE สุดท้ายนำมาใส่ที่ BYTE ของ CHECK HEADER ทุกๆ SECS จะประกอบด้วย HEADER และมีขนาด 10 BYTEE ตลอด ใน HEADER นี้จะประกอบไปด้วย

1. DEVICE ID จะบอกความหมายว่า ใคร (โดยเฉพาะจะใช้บอก ID เครื่องหรืออุปกรณ์ที่ต้องการ ติดต่อ HOST) โดย BIT แรก(ได้ชื่อว่า R BIT) จะเป็น BIT ที่ตั้งมาตรฐานไว้ว่า เป็น BIT ที่บอกว่า จาก HOST หรือ จาก EQUITMENT ที่เป็นผู้ส่ง SECS นี้ออกมา และทั้งหมดของ DEVICE ID มีความยาว 2 BYTE ซึ่งผู้ใช้งานจะได้รับทราบว่าเกิดการติดต่อระหว่างใครในระบบสื่อสาร เช่น จาก HOST CONPUTE ไปยัง EQUITMENT มายัง HOST CONPUT

2. DTEAM & FUNCTION จะบอกความหมายว่า อะไร (WHAT) หรือ BYTE ที่ใช้บอกความหมาย กิจกรรมที่จะกระทำ สำหรับที่ส่ง SECS นี้ออก STEAM และ FUNCTION จะส่งออกมาคู่กันเสมอโดย STEM จะเป็นหัวข้อใหญ่ และ FUNCTION เป็นหัวย่อย หรือจะบอกว่า STEAM เป็นชื่อกลุ่มก็ได้ เช่น S1F1 และ S1F1 จะเป็นกลุ่มการทำงานเกี่ยวกับการ CHECK READY โดย S1F1 จะเป็นการถามจาก HOST COMPUT ไปยัง EQUITMENT โดยใช้คำถามว่าคุณอยู่ใน LINE การสื่อสารนี้หรือไม่ (ARE YOU THERE REQUEST) เมื่อทาง EQUITMENT ได้รับ REQUEST ก็จะตอบกับมาว่า มีอยู่และบอกชนิดของเครื่องหรือรุ่นของเครื่องมาใช้ พร้อมกับ SOFWARE VERSION ที่ใช้งานอยู่ ซึ่งจะเห็นว่า STEAM ใช้ในการแบ่งออกเป็นกลุ่มใหญ่ BLOCK NUMBER จะบอกความหมายว่า (WHICH) โดยจะมี BIT สูงสุดเป็นตัวบอกว่าเป็น BLOCK สุดท้ายหรือยัง และถัดไปอีก 2 BIT สำรองไว้สำหรับ CHECK ERROR SYSTEM BYTE สำรองไว้สำหรับ - "BOOKKEEPING" ข่าวสารของ MESSAGE

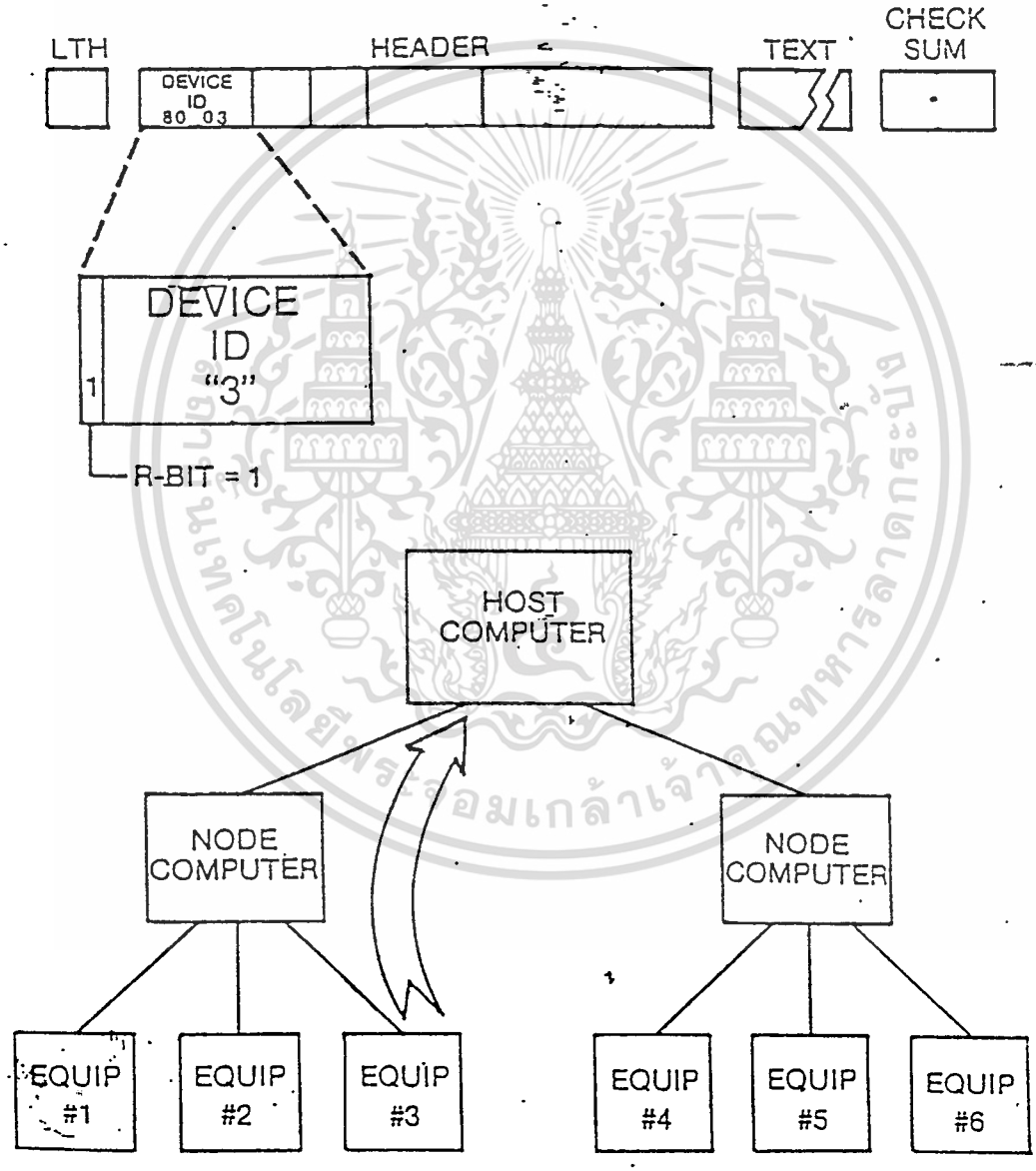
DEVICE ID

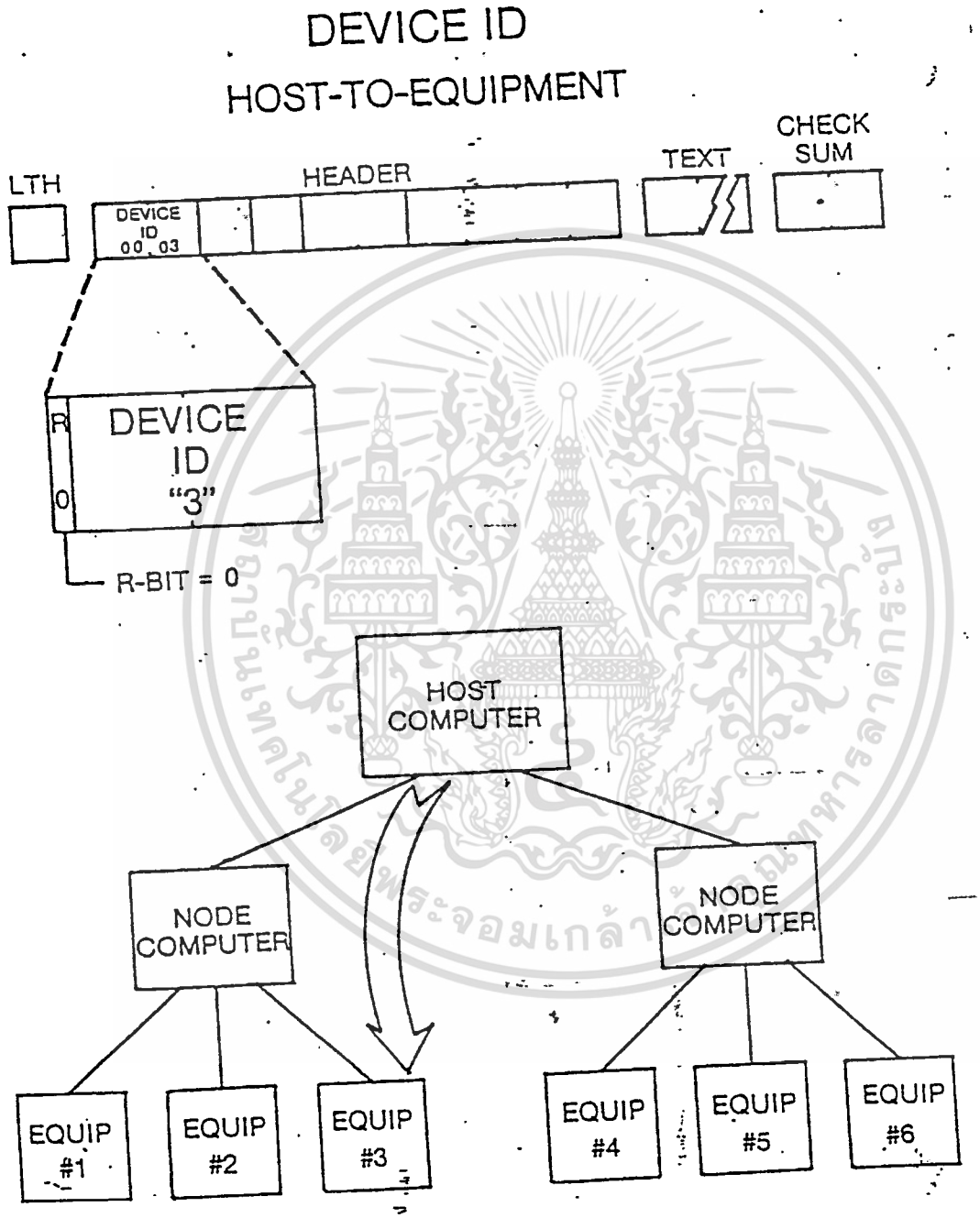


R-BIT	DEVICE ID	MESSAGE DIRECTION
0	DESTINATION	HOST TO EQUIP
1	SOURCE	EQUIP TO HOST

DEVICE ID

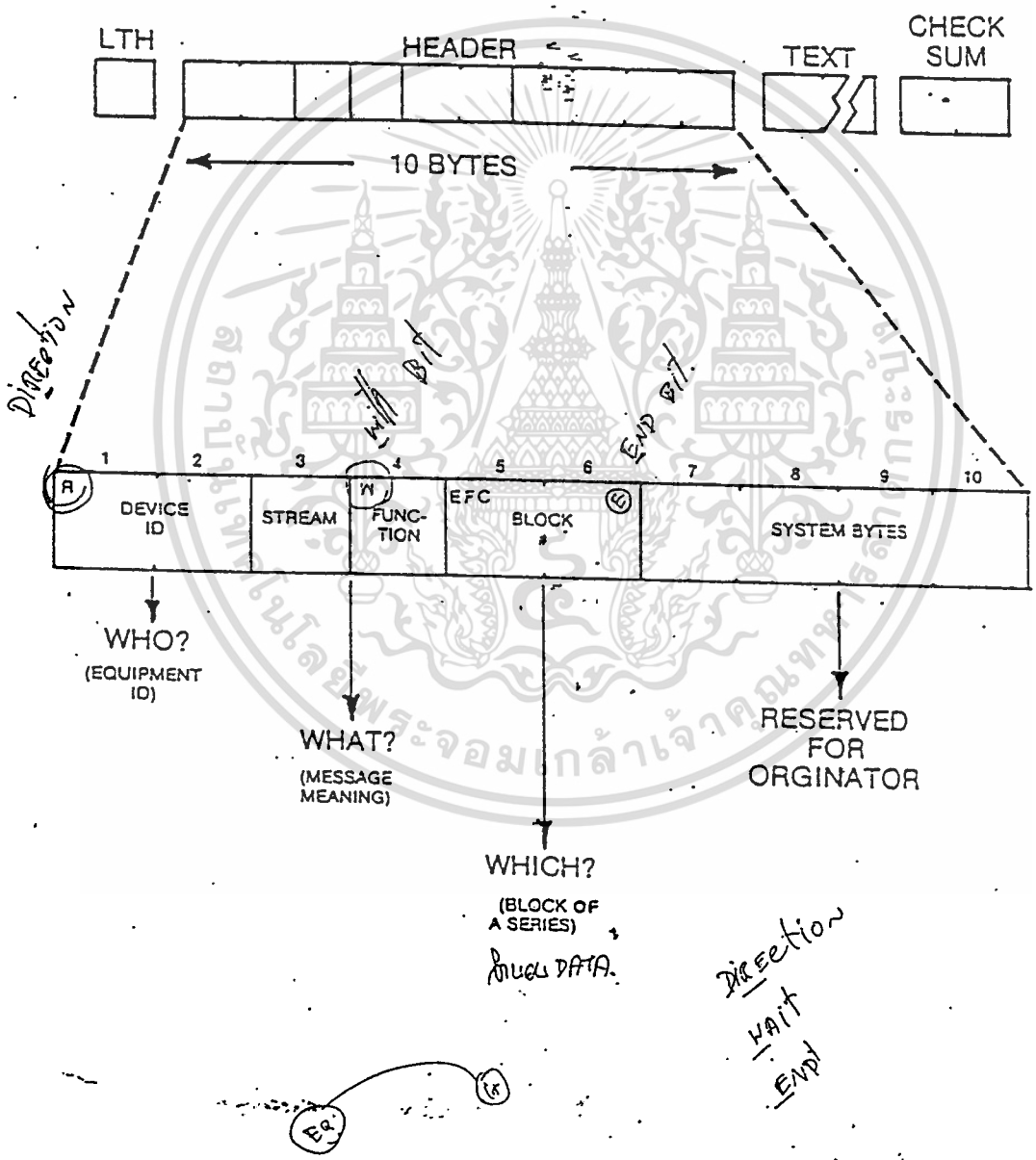
EQUIPMENT-TO-HOST





รูปที่ 14 รูปแบบช่อง BLOCK ข้อมูลเบอร์เครื่อง

SECS MESSAGE HEADER



STREAM & FUNCTION CODES

STREAM จะอยู่ตำแหน่ง BYTE ที่ 3 ของ HEADER ส่วน FUNCTION อยู่ตำแหน่งที่ 4 ใน BYTE ที่ 3 จะใช้งานเฉพาะ บิทที่ 0-6 ส่วนบิทที่ 7 ใช้ในกรณีที่ต้องการให้ผู้รับมีการตอบกลับมายังต้นทาง

STREAM — THIS DEFINES THE MAJOR TOPIC OF THE MESSAGE. THE FOLLOWING STREAM VALUES ARE USED:

0	RESERVED
1-11	DEFINED IN SECS-2
12-36	RESERVED FOR SECS-2 FUTURE USE.
64-127	AVAILABLE FOR USER-DEFINED STREAMS.

FUNCTION — THIS DEFINES THE SUB-TOPIC WITHIN STREAM. THE SAME FUNCTION COD MAY HAVE DIFFERENT MEANINGS IN DIFFERENT STREAMS, BUT THE FOLLOWING VALUES ARE USED:

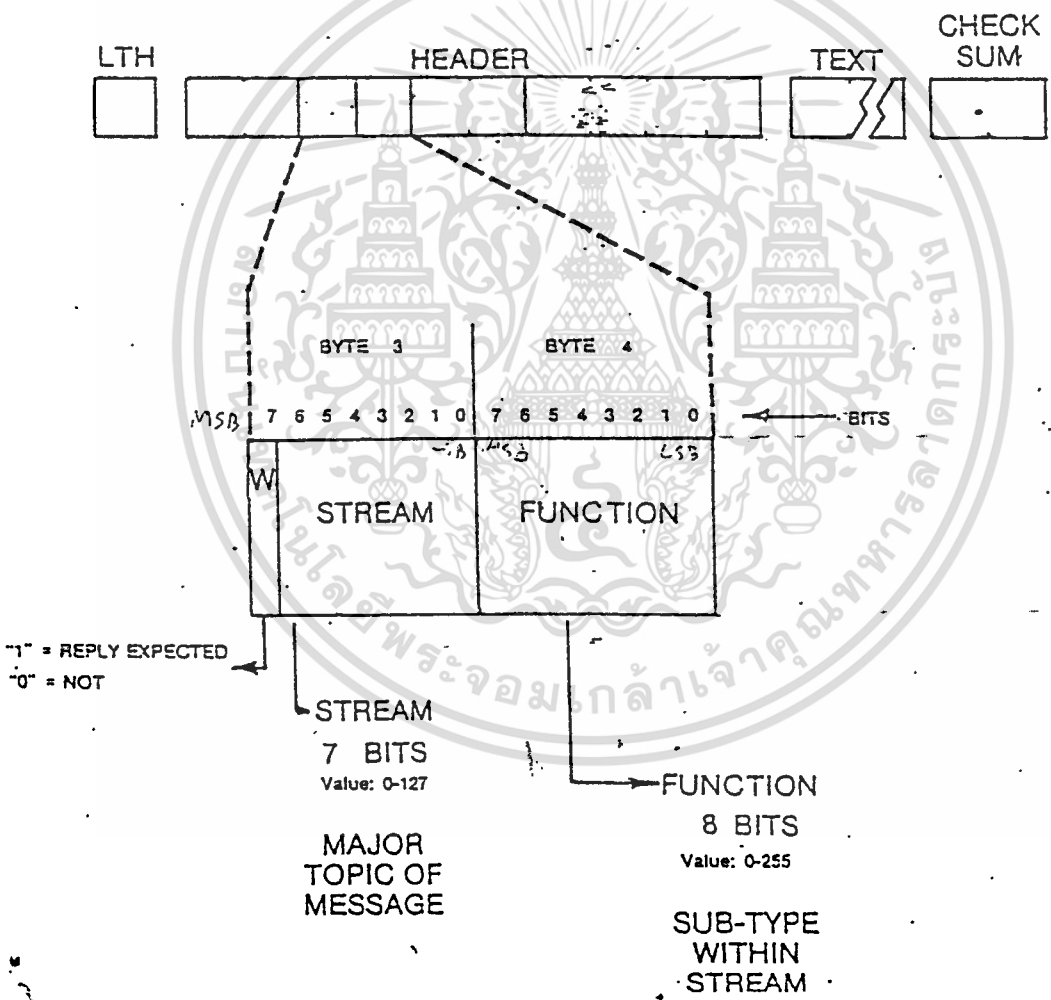
0-63	DEFINED IN SECS-2 OR RESERVED FOR FUTURE USE.
64-255	AVAILABLE FOR USER-DEFINED FUNCTIONS.

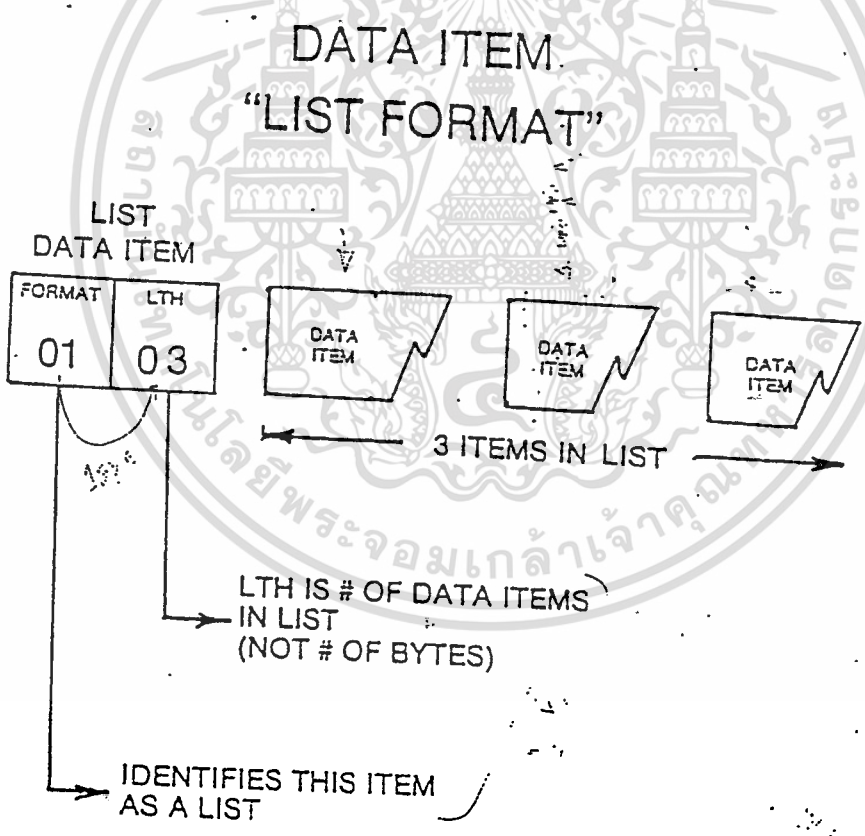
WHEN A QUREY/RESPONSE XONVERSATION IS USED, THE QUERY MESSAGE ALWAYS HAS AN ODD FUNCTION VALUE. THE ASSOCIATED RESPONSE MESSAGE HAS AN EVEN FUNCTION VALUE,ONE GREATER THE QUERY.

W-BIT — THIS BIT DEFINES WHETHER A REPLY IS EXPECTED. IF W-BIT IS "1" A REPLY IS EXPECTED. IF "0" ON REPLY IS EXPECT

SECS 2

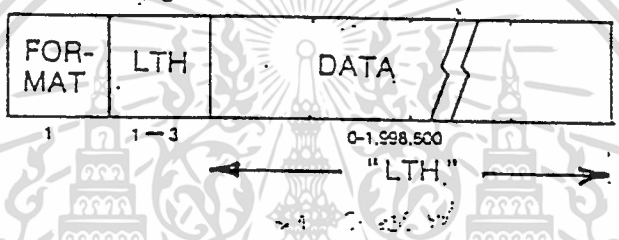
STREAM & FUNCTION CODES




DATA ITEM
"LIST FORMAT"
LIST
DATA ITEM
FORMAT
LTH
01
03
DATA ITEM
DATA ITEM
DATA ITEM
3 ITEMS IN LIST
LTH IS # OF DATA ITEMS IN LIST (NOT # OF BYTES)
IDENTIFIES THIS ITEM AS A LIST

SECS 2

DATA ITEM FORMAT



รูปที่ 18 รูปแบบของ BLOCK ข้อมูลชนิด

BOLCK ของการบ่งบอกชนิด ของข้อมูล

จากรูปจะเป็น BLOCK ที่จะใช้ใส่ข้อมูลบอกชนิดของ ข้อมูลที่ส่งมาใน PROTOCOL นี้ดังในรูปที่18 ข้อมูล BYTE แรกจะใช้ใส่ข้อมูลเบอร์ของชนิด ข้อมูล ซึ่งมีการกำหนดดังที่จะกล่าว ต่อไป และใน 3 BIT สุดท้ายปกติ จะมีค่า 001 ทั้งนี้เป็นไปตามมาตรฐาน ตัวอย่าง เช่น หมายเลขของ ข้อมูล ASCII เป็น 41 ค่าตัวเลขใน BYTE นี้จะเท่ากับ 0 ส่วนในอีก 3 BYTE ถัดมา จะใช้ใส่เลขความยาวของข้อมูลที่ส่ง ซึ่งเต็มที่ จะส่งได้เท่ากับ 1,990,600 BYTE และถ้าต้องการส่งมากกว่านี้ จะใช้ ส่งแบบ LIST (ต่อเนื่อง)

ASCII FORMAT 41 (HEX). IN THIS FORMAT, EACH BYTE OF DATA CONTAINS AN ASII CHARACTER.

UNSIGNED BINARY FORMAT 21 (HEX). IN THIS FORMAT, EACH BYE OF DATA

OF OTHER DATA WHICH USES ALL 8 BIT OF THE BYTE. A PARTICULAR CASE OF UNSIGNED BIARY IS THE ONE-BYTE UNISIGNED BINARY FIELD. THIS IS WIDELY USED IN THE SECS-2 FORMATS TO HOLD A VALUE IN THE - FANGE 0 THROUGH 255

INTEGER

IN THIS FORMAT, DATA CONTAINS A SIGNED,FIXED-POINT, INTERGER BINARY VALUE. THE SIGN IS ALWAYS THE - HIGHEST BIT, WITH "0" SIGNIFYING POSITIVE (OR ZERO) AND "1" SIGNIFYING NEGATIVE VALUES. NEGATIVE VALUES ARE CARRIED IN TWOS-COMPLEMENT FORM.FOR DATA LONGER THAN 1 BYTE, THE MOST SIGNIFICANT BYTE (MSB) IS ALWAYS FIRST. AND THE LEAST-SIGNIFICANT BYTE (LSB) IS ALWAYS LAST.

THERE ARE FOUR (4) FORMATS OF INTEGER DEFINED.

FORMAT 65 (HEX). ONE-BYTE INTEGER. THE VALUE IN DATA MAY RANGE FROM -128 TO + 127.

FORMAT 69 (HEX). TWO-BYTE INTEGER. THE VALUE IN DATA MAY FROM -32,768 TO +32,767.

FORMAT 71 (HEX). FOUR-BYTE INTEGER. THE VALUE IN DATA MAY RANGE FROM -(2**31)-1).

FORMAT 61 (HEX). EIGHT-BYTE. THE VALUE IN DATA MAY

FLOATING-POINT THIS FORMAT USES THE IEEE P754 FORMAT FOR FLOATING-POINT VALUES. THERE ARE TWO (2) FORMATS OF FLOAT ING-POINT DEFINED.

FORMAT 91(HEX). FOUR-BYTE FLOATING POINT.

FORMAT 81(HEX). EIGHT-BYTE FLOATING POINT.

ZERO LENGTH DATA ITEM

FOR-MAT	LTH
	00

CAN ACT AS A "PLACE HOLDER"

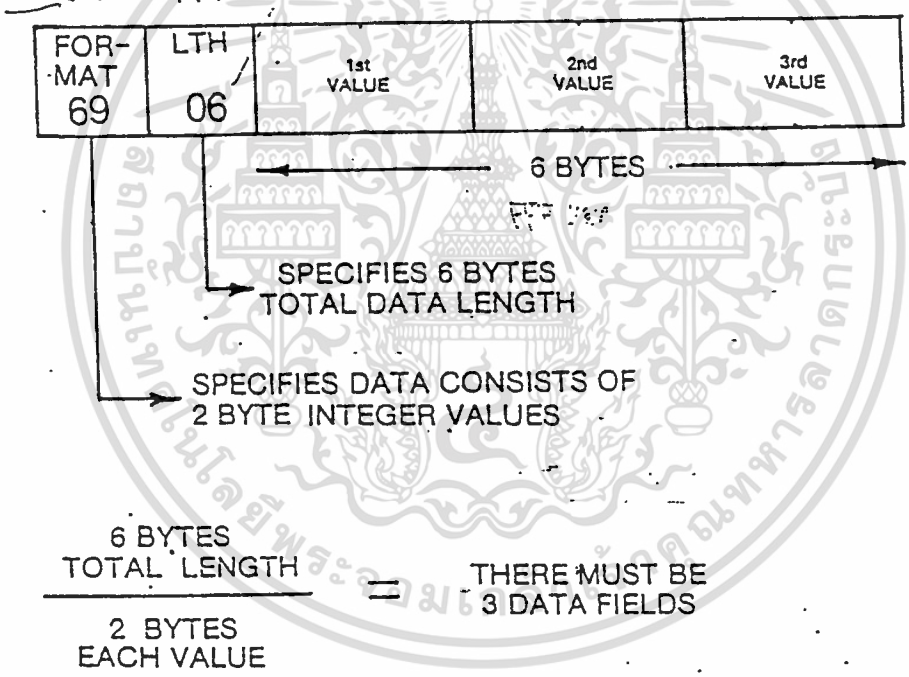
ZERO-LENGTH DATA ITEM

ALTHOUGH IT MIGHT NOT BE IMMEDIATELY APPARENT, THERE IS AN IMPOUTANT USE FOR A DATA ITEM WITH A LENGTH OF ZERO (0). SUCH A DATA ITEM OFTEN FUNCTIONS AS A "PLACE HOLDER". AS A SIMPLIFIED EXAMPLE, IMAGINE A "NAME" RECORD, WITH THREE DATA ITEMS:

FIRST NAME
MIDDLE NAME
LAST NAME

FOR A PERSON WHO HAD NO MIKKLE NAME, THE "MIDDLE NAMEK" DATA COULD USE A LENGTH OF ZERO. THIS KEEPS THINGS STRAIGHT, SO THAT THE LAST NAME IS ALWAYS THE THIRD DATA ITEM.

MULTIPLE SUB-ITEMS IN ONE ITEM



NOTE: THIS IS NOT THE SAME AS A LIST.

LIST DATA ITEMS

AS SHOWN IN THE DIAGRAM, FORMAT CODE 01(HEX) SPECIFIES A LIST ITEM.

A LIST DATA ITEM CONSISTS OF A SERIES OF OTHER DATA ITEMS. FOR A LIST DATA ITEM, THE LENGTH FIELD FIELD SPEDIFIES THE NUMBER OF DATA ITEMS CONTAINED IN THE LIST, NOT THE NUMBER OF BYTES IN THE LIST.

TO DETERMINE THE LENGTH IN BYTES OF A LIST, YOU MUST EXAMINE THE INDIVIDUAL DATA ITEMS CONTAINED WITHIN THE LIST.

LIST ITEMS CAN BE EMBEDDED WITHIN OTHER LIST ITEMS.

ตัวอย่างการส่งข้อมูล จาก ID XXXXXXXXXXXXXXX STREAM 1 FUNCTION 2 และ ตาม ด้วย ข้อมูล รุ่นของ EQUIPMENT และ โปรแกรม VERSION

00011100	BLOCK LENGTH 28 BYTES
1XXXXXXX	R=1(TO THE HOST)
XXXXXXXX	DEVICE ID.
00000001	STREAM 1, W=0.
00000010	FUNCTION 2.
10000000	E=1
00000001	BLOCK # 1.
XXXXXXXX	SYSTEM BYTES.
XXXXXXXX	SYSTEM BYTES.
XXXXXXXX	SYSTEM BYTES.
XXXXXXXX	SYSTEM BYTES.
00000001	LIST
00000010	2 LIST ELEMENTS
01000001	MDLN FORMAT ASCII, 1 LENGTH BYTE
00000110	MDLN LENGTH 6 BYTES
01000001	A
01000011	C
01001101	M
01000101	E
00100000	BLANK
00100000	BLANK
01000001	SOFTREV FORMAT ASCII, 1 LENGTH BYTE
00000110	SOFTREV LENGTH 6 BYTES
00110000	0

00110000	0
00110000	0
00110000	0
00110001	1
XXXXXXXX	CHECKSUM MSB
XXXXXXXX	CHECKSUM LSB

บทที่ 5

5.1 VGBL SOFTWARE

VGBL คือ ซอฟท์แวร์สื่อสาร (COMMUNICATION SOFTWARE PACKAGE)ซึ่งถูกพัฒนาขึ้นใช้กับบอร์ด (BOARD) GEIC-32 ซึ่งช่วยให้การเชื่อมโยงไปยัง HOST และอุปกรณ์เป็นไปได้อย่างรวดเร็ว และสามารถเพิ่มจำนวนอุปกรณ์ที่ต่อกับ HOST ได้ การจัดการข่าวสารบางอย่างใน VGBL จะทำให้การตอบสนองข่าวสารทำได้เร็วขึ้นและสามารถลดเวลาที่ไม่จำเป็นใน CPU

VGBL จะประกอบด้วย

* PXI - GEIC - VAX โมดูลสื่อสาร (COMMUNICATION MODULE) และโมดูลใดโมดูลหนึ่งข้างล่างนี้คือ
* XPC (MULTI-PORT SECS COMMUNICATION)
* GEIC โปรแกรมใช้งาน (APPLICATION PROGRAM)

PXI คือ โมดูลสื่อสารร่วม ซึ่งอนุญาติให้ GEIC ติดต่อกับ VSII-VAX ซอฟ์ทแวร์สื่อสารได้

XPC คือ โมดูลสื่อสารแบบมัลติพอร์ต ซึ่งทำให้ GEIC สามารถติดต่อกับอุปกรณ์ในการผลิตที่สามารถสนทนา SECS ได้

GEIC โปรแกรมใช้งานสำหรับการเก็บรวบรวมข้อมูล สามารถใช้มาตรฐาน PXI โมดูลเพื่อการถ่ายโอนข้อมูลไปยังระบบ VAX

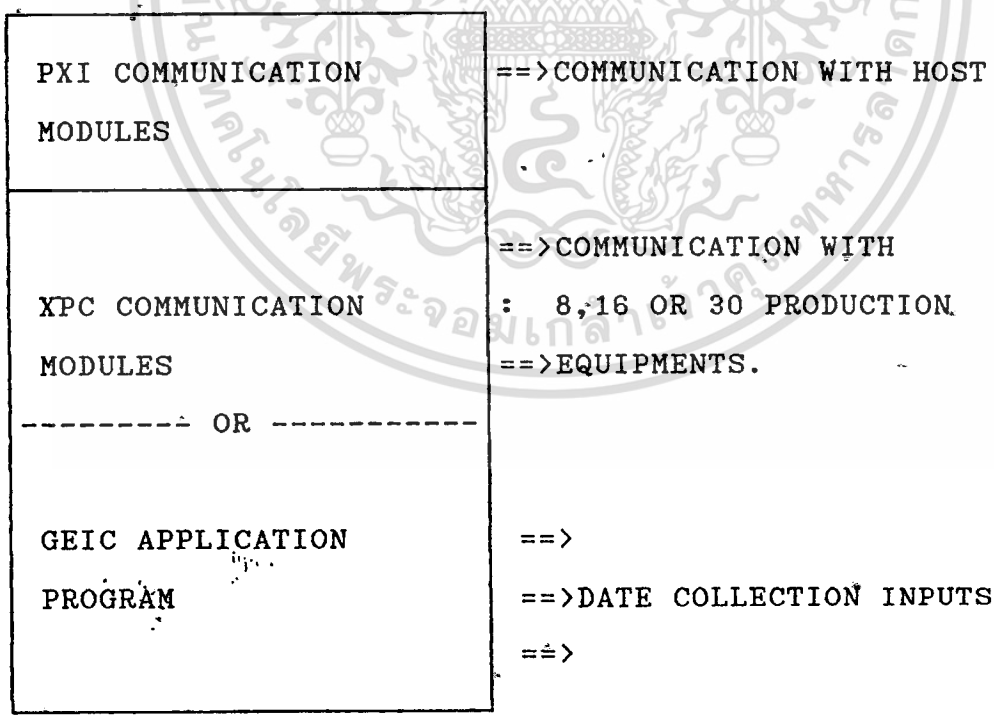
การออกแบบโครงสร้าง

* ติดต่อกับอุปกรณ์ที่ใช้มาตรฐาน SEC (SEMICONDUCTOR EQUIPMENT COMMUNICATION
* ติดต่อกับ HOST โดยใช้โปรโตคอล (PROTOCOL) PXI
* สามารถออกแบบให้จัดการกับข่าวสาร SECS ได้ในแต่ละแห่งโดยตนเอง
* อำนวยความสะดวกในการแปลงข่าวสาร SECS/PXI
* กำหนดเส้นทางข่าวสารระหว่างอุปกรณ์และ HOST
* อำนวยความสะดวกในการเก็บข้อมูลที่สำคัญ

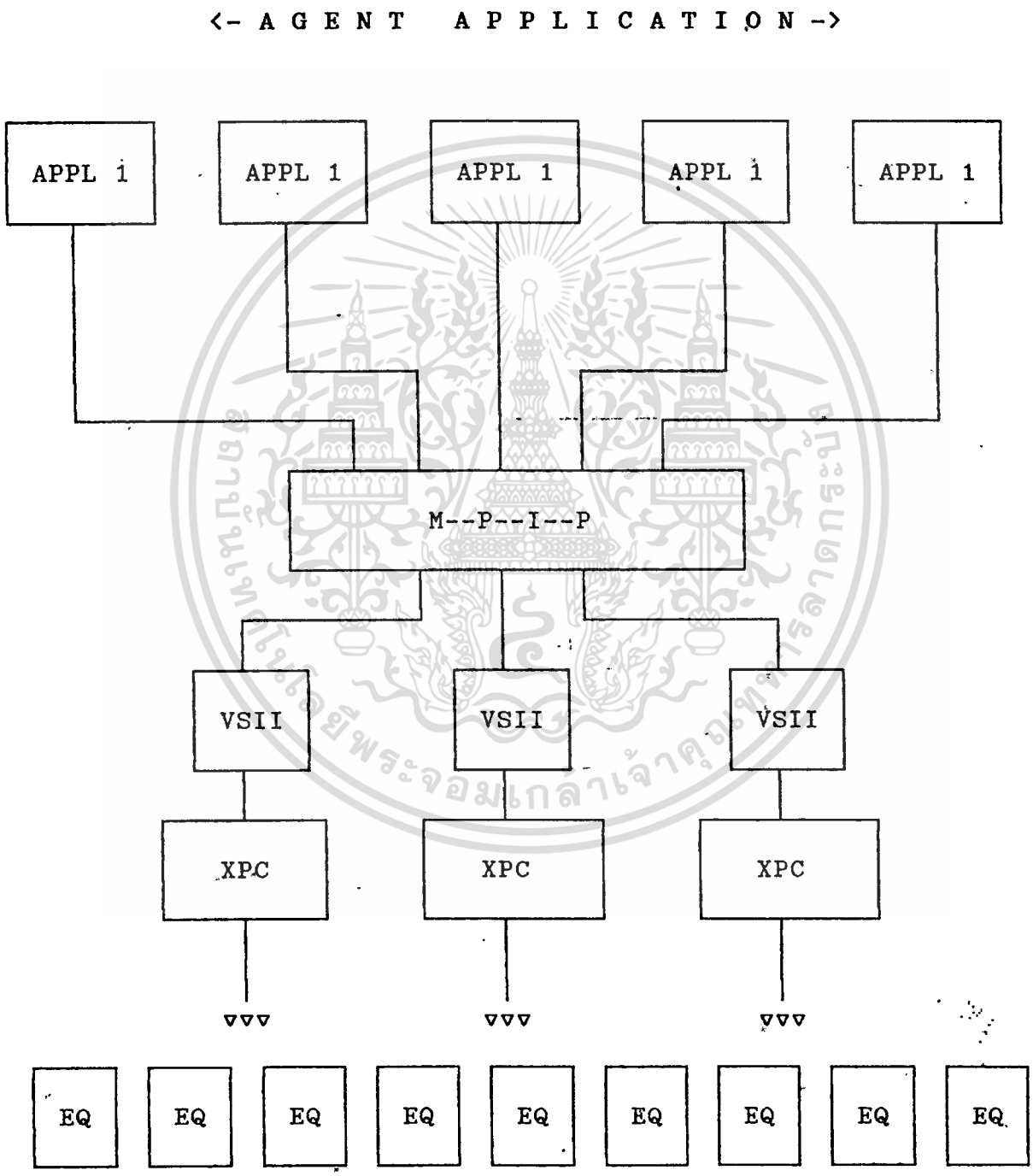
ข้อจำกัดในการออกแบบ

* ชนิดของอุปกรณ์เฉพาะอย่างจะต้องถูกกำหนดและเขียน USR (USER SERVICE ROUTINE)
* ต้องการระบบ HOST ริเริ่มการเชื่อมโยงสื่อสารในช่วงแรกที่ GEIC เริ่มทำงาน
* อุปกรณ์ใหม่ ๆ ไม่อนุญาติให้เข้าเชื่อมโยงได้ในขณะที่ข่ายของ HOST เกิดการขัดข้อง

VGBL COMFIGURATION DIAGRAM



OVERALL VSII ARCHITECTURE DIAGEAM



VGBL SOFTWARE ARCHITECTURE.

โมดูลสื่อสาร PXI ประกอบด้วย 6 TASKS และ 3 QUEUES ดังนี้คือ

	TASKS	ID : PRIORITY	
1.	PXI_PORT	30 : 9	
2.	PXI_BXFER	32 : 10	
3.	PXI_MSGRX	34 : 18	
4.	PXI_MSGTX	36 : 16	
5.	PXI_TIMER	38 : 12	
6.	PXI_MON	40 : 30	* OPTIONAL
	QUEUE	QUEUE ELEMENT	
1.	QUE_PXI_RSND	SND_COUNT	
2.	QUE_PXI_PORT	1000	
3.	QUE_PXI_MSGRX	RCV_COUNT	
4.	QUE_PXI_MSGTX	SND_COUNT	
5.	QUE_ACEB	5	
6.	QUE_SW1	2	

โมดูลสื่อสาร XPC ประกอบด้วย 3 TASKS และ 2 QUEUES ดังนี้คือ

	TASKS	ID : PRICRITY
1.	XPC_BXFER	20 : 10
2.	XPC_MSGRX	22 : 18
3.	XPC_TIMER	24 : 12
	QUEUE	QUEUE ELEMENT
1.	QUE_SEC_COMM	1000

VGBL HARDWARE REQUIPMENT.

* GEIC BOARD
* 8 CHANNEL SERIAL I/O BOARD.
* BOX ASSEMBLY
* DECSERVER LINE
* CONSOLE TERMINAL (OPTIONAL)

DEFINE SPECIFICATION SOFTWARE VGBL

* โมด (MODE) ต่าง ๆ สำหรับการ START UP VGBL/HOST

MODE 1 : เมื่อ VGBL และ HOST เพิ่มจะเริ่มทำงาน

- VGBL จะอยู่นิ่งเฉย
- รับ STARTUP REQUEST จาก HOST
- ส่ง PXI STARTUP REPLY ไปยัง HOST
- ส่ง TIME REQUEST ไปยัง HOST
- เซ็ตสภาวะ HOST = 1
- เริ่มสภาวะศูนย์ไปยังพอร์ตของอุปกรณ์
- ส่งข่าวสาร EQUIPMENT ONLINE ไปยัง HOST เสมือนเครื่องใหม่เข้ามา "ONLINE"

MODE 2 : VGBL REBLLTED, HOST ทำงานแล้ว

- VGBL ยังนิ่งอยู่
- รับลำดับ "PRESENCE POLLING" จาก HOST
- ส่ง STARTUP REQUEST ไปยัง HOST
- รับ STARTUP REPLY จาก HOST
- ส่ง TIME REQUEST ไปยัง HOST
- เซ็ตสภาวะ HOST = 1
- เริ่มสภาวะศูนย์ไปยังพอร์ตของอุปกรณ์
- ส่งข่าวสาร EQUIPMENT ONLINE ไปยัง HOST เสมือนมี

MODE 3 : VGBL ทำงานแล้วม HOST REBOOTED

- รับ STARTUP REQUEST จาก HOST
- ส่ง STARTUP REPLY ไปยัง HOST

MODE 4 : VGBL ปกป้อง HOST "OFLINE"

- เซ็ตสภาวะ HOST = 0
- กระทำ PXI RECOVER ยังข่าวสาร PXI ที่มีอยู่
- สนับสนุนอุปกรณ์ต่าง ๆ ที่ได้ต่อเชื่อมแล้วต่อไป
- ไม่อนุญาติให้อุปกรณ์ใหม่เข้ามา "ONLINE"
- พักสภาวะศูนย์ไปยังพอร์ตที่ไม่ได้ต่อไว้
- คงไว้ซึ่ง "PRESENGE POLLING"ไปยังพอร์ตที่ต่อไว้
- รอจนกระทั่ง HOST ส่ง STARTUP REQUEST

* ทำให้ข่าวสารจากอุปกรณ์ให้เกิดประโยชน์ใช้สอยดังนี้คือ

1. สนับสนุนชนิดของการปฏิบัติการ (OPERATION TYPE)
2. การติดตั้งอุปกรณ์ใหม่
3. การเปลี่ยนชื่อกำหนดอุปกรณ์ (EQUIPMENT ID)
4. สนับสนุนการทำงานตามกระแส และฟังก์ชั่น (FUNCTION)
   (อ้างอิง APPENDIX E สำหรับ VALIDATION ALGORITHM)

1. ป้องกันชนิดอุปกรณ์ที่ไม่ค้ำจุนกับ (UNSUPPORTED EQUIPMENT)
   - รายงานไปยัง HOST เมื่อได้รับข่าวสารจากอุปกรณ์ชนิดที่ไม่ค้ำจุนกันจากพอร์ต
   - ปฏิบัติตามลำดับการทำงานของอุปกรณ์ภายนอก
2. ป้องกันการติดตั้งอุปกรณ์ใหม่
   - ตรวจสอบความถูกต้องและกำหนดชื่ออุปกรณ์ซ้ำกัน
   - ถ้าการกำหนดชื่อไม่ถูกต้องหรือซ้ำกันเกิดขึ้น
     * รายงานไปยัง HOST ว่าไม่ถูกต้องหรือมีการซ้ำกัน
     * ปฏิบัติตามลำดับขั้นตอนของอุปกรณ์ภายนอก (OFLINE EQUIPMENT)
   - ถ้าไม่มีปัญหาการกำหนดชื่ออุปกรณ์

* แสดงพอร์ตให้เป็น "ACTIVE"
* แสดงจำนวนการเลือกพอร์ต

3. ป้องกันการเปลี่ยนชื่อกำหนดของอุปกรณ์
   - รายงานไปยัง HOST ว่าอุปกรณ์ปัจจุบันนั้นเป็น "OFLINE"
   - ปฏิบัติตามลำดับขั้นตอนของอุปกรณ์ "OFLINE"

4. เมื่อรับขั้นตอนปฏิบัติที่ไม่ค้ำจุนจากอุปกรณ์
   - รายงานไปยัง HOST ว่าได้รับข่าวสารที่ไม่ค้ำจุนกันจากอุปกรณ์
   - ขจัดข่าวสารนั้นทิ้งไป

* VGBL ป้องกันอุปกรณ์ไปยัง "OFLINE"
   - รายงานไปยัง HOST ของอุปกรณ์ "OFLINE"
   - สะสางข่าวสารทั้งหมดที่มีอยู่จากอุปกรณ์
   - ทำการเลือกศูนย์จากพอร์ต

* ข่าวสารทั้งหมดจากอุปกรณ์จะถูกส่งผ่านไปยัง USER SERVICE ROUTINE หลังจากวิธีการตรวจสอบความถูกต้องเรียบร้อยแล้ว

* ทำให้ข่าวสารทั้งหมดจาก HOST เกิดประโยชน์ใช้สอยดังนี้คือ
   1. สนับสนุนชนิดของการปฏิบัติการ
   2. สนับสนุน GEIC ID
   3. สนับสนุน PIP ID
   4. สนับสนุนการกำหนดชื่ออุปกรณ์
   5. สนับสนุนการทำงานตามกระแสและฟังก์ชั่น (FUNCTION)
      (อ้างอิง APPENDIX E สำหรับ VALIDATION ALGORITHM)

1. ป้องกันชนิดที่ไม่ค้ำจุนกัน (UNSUPPORTED)
   - แจ้งให้ HOST ทราบถึงความคลาดเคลื่อน
   - เซ็ทสภาวะของ HOST ไป 0
   - รอรับ STARTUP REQUEST จะ HOST

2. ป้องกัน GEIC ID ที่ไม่ค้ำจุนกันหรือเท่ากับ 0

* กำหนดสภาวะ HOST = 1
* ส่ง STARTUP REPLY ไปยัง HOST

หรือ กระทำตามขั้นตอนในข้อ 1.

3. ป้องกัน PIP ID เปลี่ยนไป
   - เป็นไปตามขั้นตอนในข้อ 1.
4. ป้องกันข่าวสารจากอุปกรณ์ที่ไม่ค้ำจุนกัน
   - แจ้งให้ HOST ทราบถึงความผิดพลาด
   - ขจัดข่าวสารนั้นทิ้งเสีย
5. ป้องกันข่าวสารที่ไม่ค้ำจุนกัน
   - เช่นเดียวกับข้อ 4.

* VGBL รับ DUPLICATE EQUIPMENT ID จาก HOST
   - รีเซ็ตพอร์ตอุปกรณ์ให้ว่าง (IDLE)
   - สะสางข่าวสารที่มีอยู่จากอุปกรณ์
   - ทำสภาวะศูนย์ไปยังพอร์ตทั้งหลาย

สิ่งที่ HOST ต้องกระทำในสภาวะต่อไปนี้คือ

* รับข่าวสารและตรวจสอบถ้าเป็นข่าวสาร RESEND/SPOOL แล้วจะต้อง
   - ตอบรับด้วย EQID เซ็ตเป็น 0
   - แสดงข่าวสารไปยังคอนโซล (CONSOLE)
* รับข่าวสาร EQUIPMENT ONLINE
   - กระทำการตรวจสอบ EQID
   - กำหนด EQID ในตาราง ROUTE
   - แสดงข่าวสารไปยังคอนโซล
* รับข่าวสารจาก GEIC
   - กระทำการตรวจสอบ
   - แสดงข่าวสารที่ได้รับไปยังคอนโซล
* รับข่าวสาร EQUIPMENT OFLINE
   - ย้าย EQID จากตาราง ROUTE

* ป้องกัน GEIC OFLINE
  - สะสางอุปกรณ์ที่ต่อเชื่อมไปยัง GEIC
  - แสดงข่าวสารไปยังคอนโซล
  - เริ่มขบวนการ STARTUP
* ป้องกันการเปลี่ยน GEIC ID หรือ DUPLICATE GEIC ID
  - สะสางอุปกรณ์ที่ต่อไปยัง GEIC
  - ส่ง S159F1 ไปยัง GEIC
  - แสดงความผิดพลาดไปยังคอนโซล
  - เริ่มขบวนการ STARTUP
* ป้องกัน DUPLICATE EQID ในข่าวสาร EQUIPMENT ONLINE
  - ขจัดข่าวสารทิ้ง
  - ส่ง S159F165 ไปยัง GEIC
  - แสดงความผิดพลาดไปยังคอนโซล
* ป้องกันข่าวสารที่ไม่ค้ำจุนกันกับ PIPD ,GID หรือ TYPE
  - สะสางการเชื่อมโยงอุปกรณ์ไปยัง GEIC
  - ส่ง S159F1 ไปยัง GEIC
  - แสดงความผิดพลาดไปยังคอนโซล
  - เริ่มขบวนการ STARTUP
* ป้องกันข่าวสารที่ไม่สค้ำจุนกันกับ EQUIPMENT ID
  - ขจัดข่าวสารทิ้ง
  - ส่ง S159F65 ไปยัง GEIC
  - แสดงความผิดพลาดไปยังคอนโซล

STANDARD MESSAGES SENT FROM VGBL TO PIP (HOST).

* VGBL STARTUP REQUEST.	S151F1	[R]
* REQUEST FOR TIME.	S152F17	[R]
* REPORT EQUIPMENT ID LIST.	S151F165	

* REPORT EQUIPMENT "OFLINE"	S151F169	
* UNSUPPORTED PIPID,GID,TYPE-	S159F1	[*]
* UNSUPPOTED STREAM	S159F3	
* UNSUPPORTED FUNCTION	S159F5	
* UNSUPPORTED EQUIPMMENT	S159F65	

THE FOLLOWING REPORTS ACTIVITY BETWEEN EQUIPMENT AND GEIC

* REPORT UNRECOGNISED TYPE.	S159F101	[X]
* REPORT UNRECOGNISED STREAM	S159F103	
* REPORT UNRECOGNISED FUNCTION	S159F105	
* GEIC DETECT DUPLICATE EQID	S159F165	[X]

STANDARD MESSAGES SENT FROM PIP (HOST) TO VGBL.

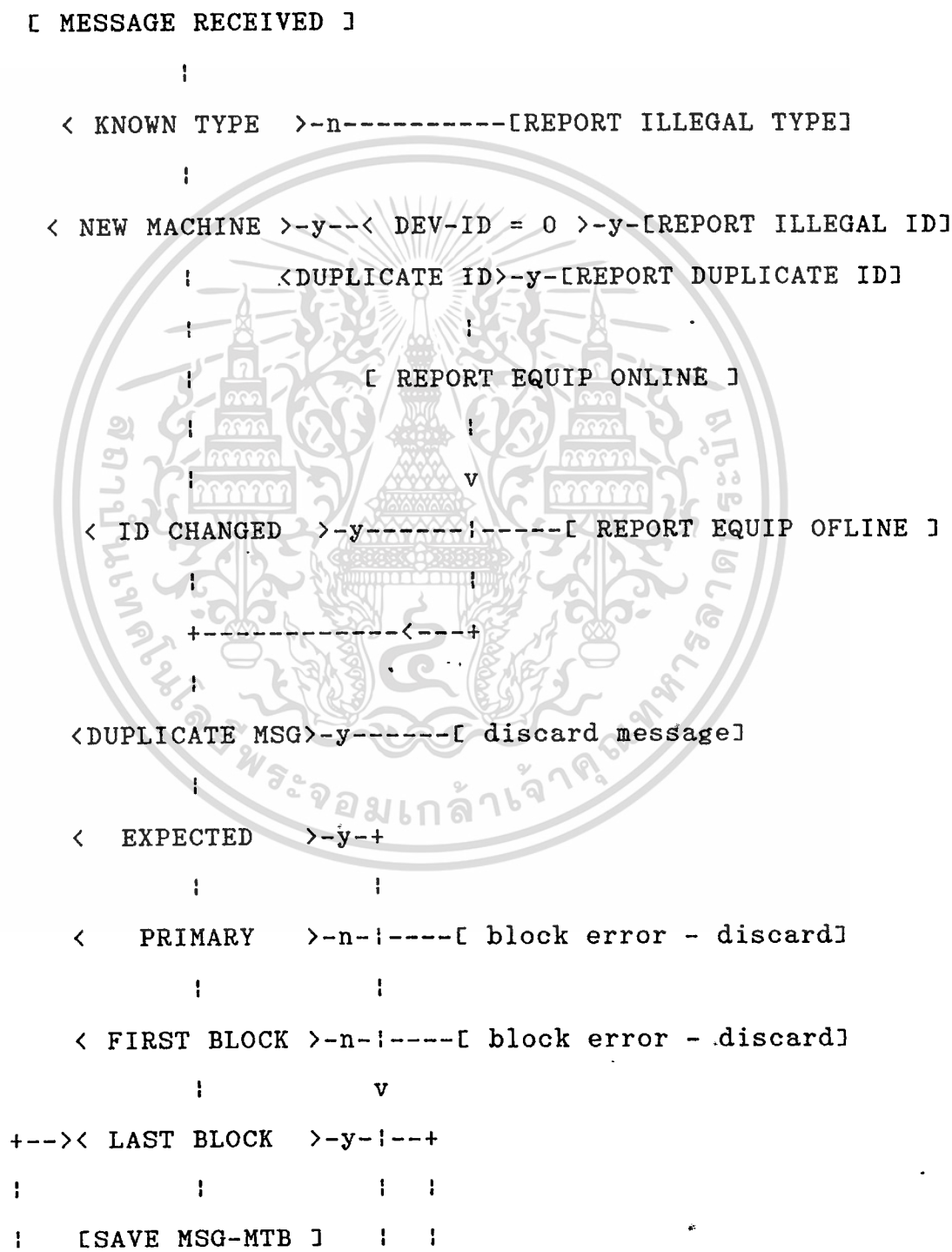
* VSII STARTUP REQUEST	S151F1	[R]
* UNSUPPORTED PIPID,GID OR TYPE	S159F1	[*]
* UNSUPPORTED STREAM	S159F3	
* UNSUPPORTED FUNCTION	S159F5	
* UNSUPPORTED EQUIPMENT	S159F65	
* HOST DETECT DUPLICATE EQID	S159F167	

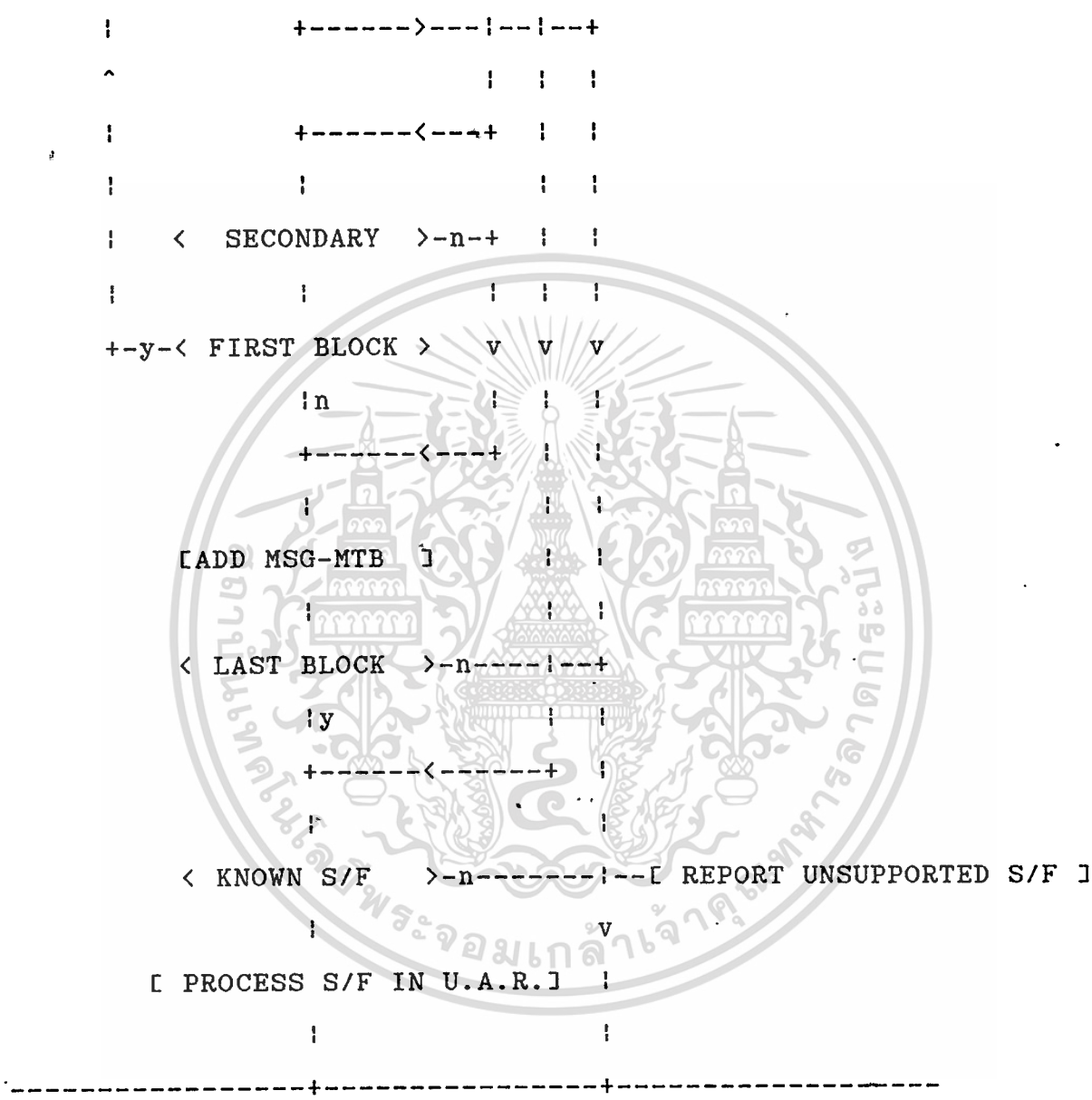
NOTE : [*] SET XPC TO "OFLINE"

[R] MESSAGE REPLY REQUIRED.

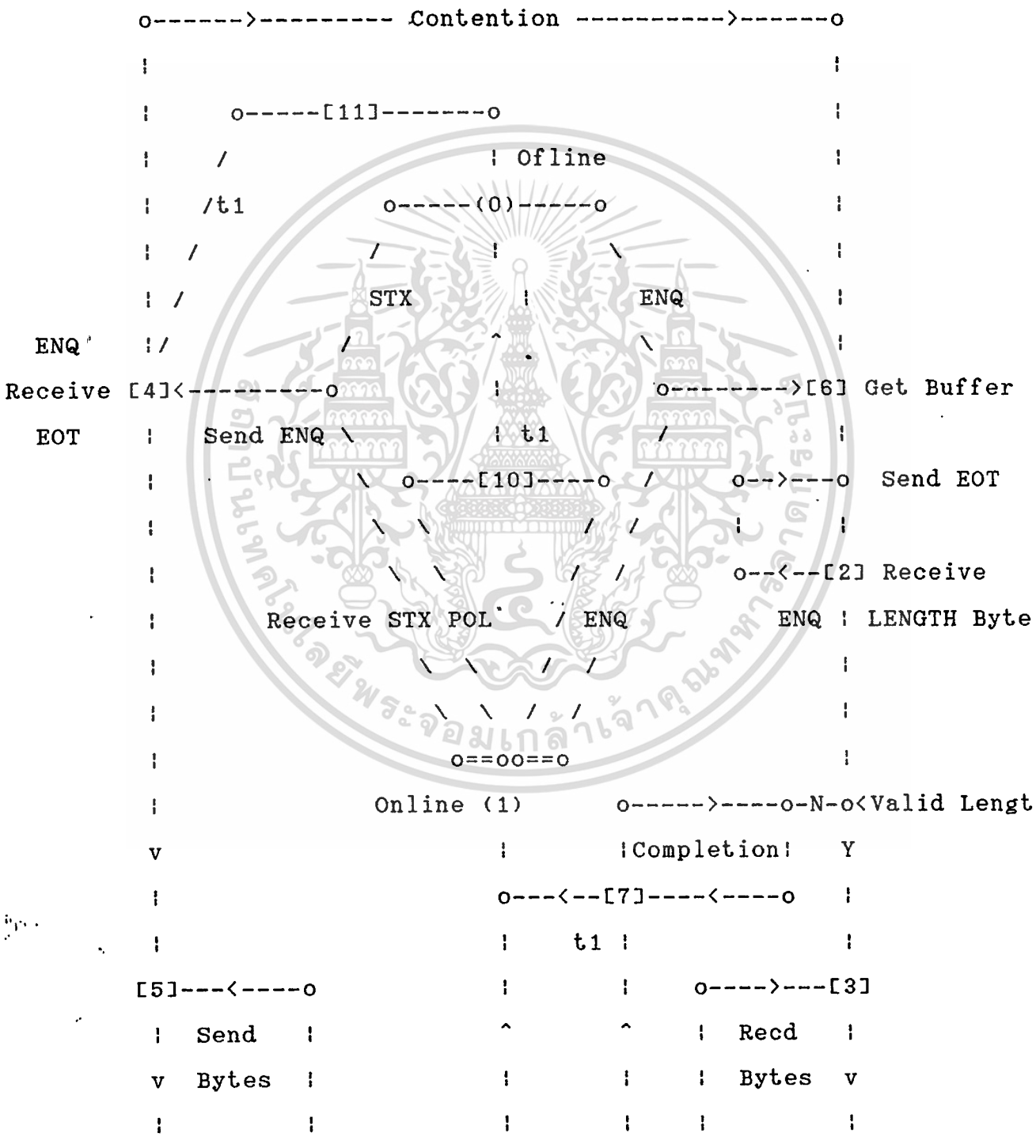
[X] SET PORT TO "ONLINE"

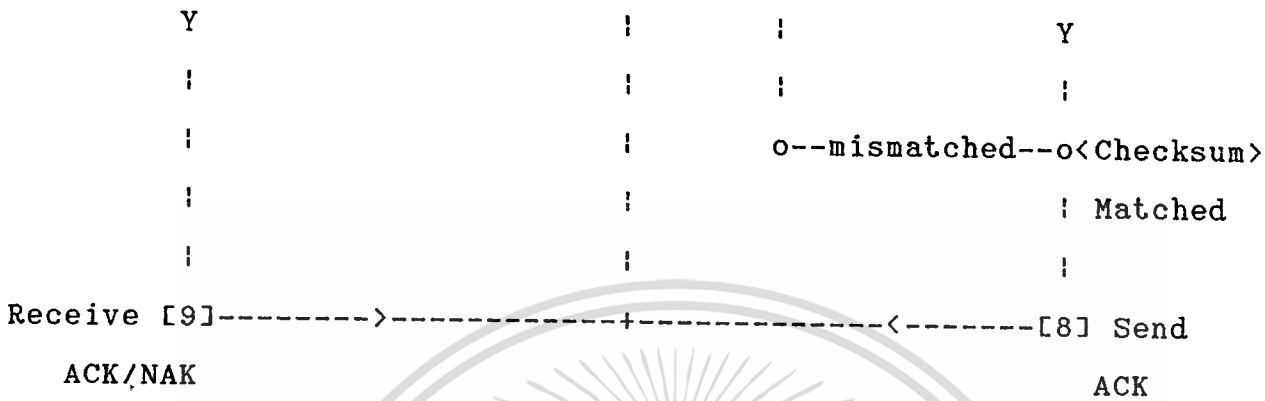
SECS MESSAGE VALIDTION ALGORITHM




SECONDARY
FIRST BLOCK
ADD MSG-MTB
LAST BLOCK
KNOWN S/F
REPORT UNSUPPORTED S/F
PROCESS S/F IN U.A.R.
wait for the next block

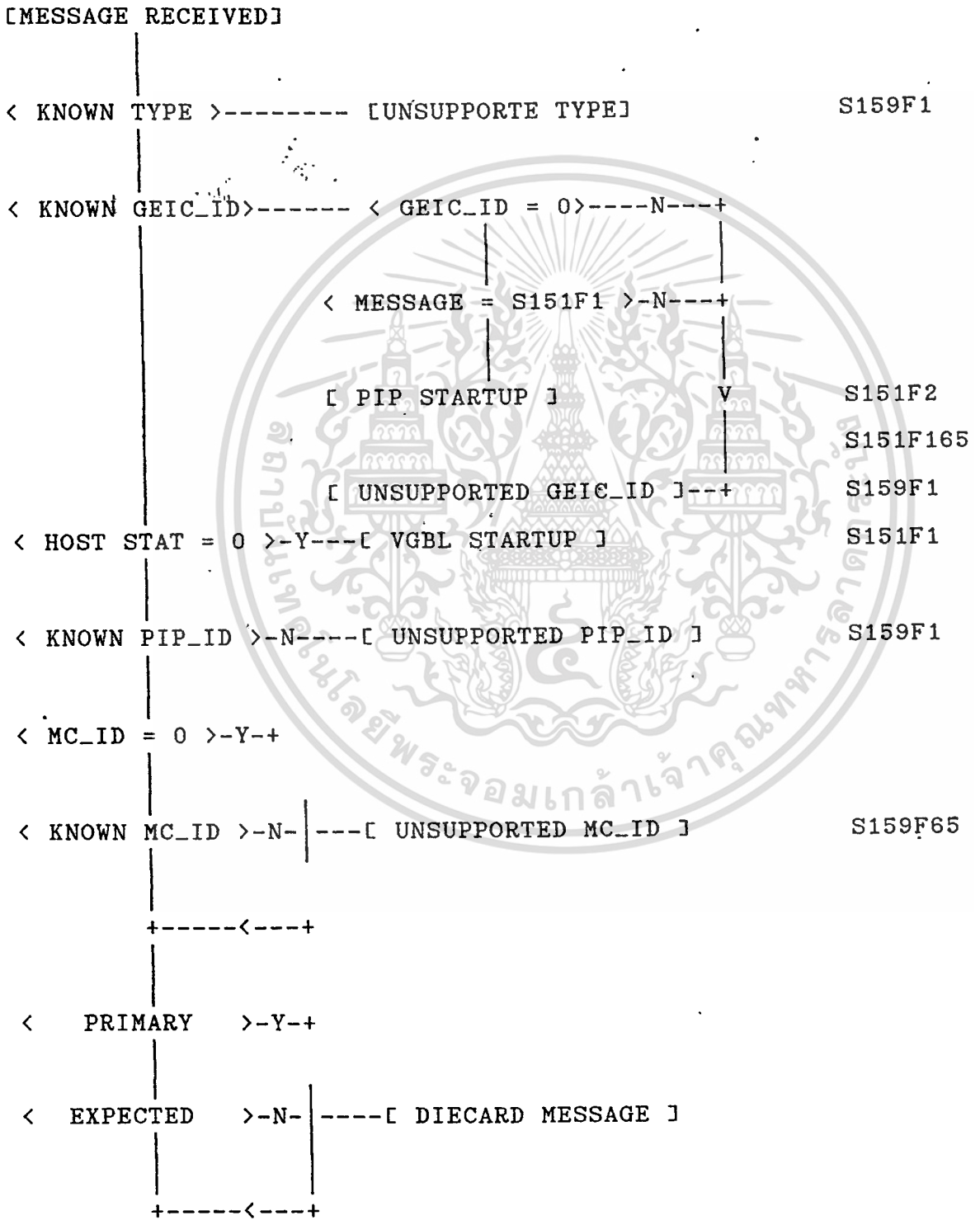
BLOCK TRANSFER COMMUNICATION STATE.





IMPERFECT WORLD

PXI MESSAGE VALIDATION ALGORITHM



[UNSUPPORTED FUNCTION] S159F5

[PROCESS S/F IN U.S.R.]

SYSTEM SUPPORTED S/F

S151F1 / S151F2

S152F25 / S152F26

S159F167

CONSOL TERMINAL

T1,T2,T3,T4 บนจอ PROG-DIAGRAM คืออะไร

T1 TO T4 คือ ตัวนับ (COUNTER) ซึ่งแสดงจำนวนครั้งที่ SECS MESSAGE มี TIME OUT สำหรับสถานะการณ์ตามนี้

1. T1 (RECEIVE TIMEOUT): GEIC TIMEOUT ขณะที่รอ BYTE ถัดไปของ SECS MESSAGE อันอาจจะเนื่องจาก
   1) อุปกรณ์หยุดการส่งทันทีใดทั้งที่ยังส่งข่าวสารไม่หมด
   2) ข่ายสื่อสารเชื่อมโยงระหว่าง GEIC และอุปกรณ์ถูกตัดขาด
2. T2 (PROTOCOL TIMEOUT): ตัวนับนี้จะแสดงจำนวนครั้งที่ GEIC ส่ง ENQ ไปยังอุปกรณ์ และไม่ได้รับการตอบสนองจากอุปกรณ์ (TIMEOUT จะรอ EOT จากอุปกรณ์) หรือเมื่อ GEIC TIMEOUT รอรับ ACK หรือ NAK จากอุปกรณ์หลังจาก GEIC ส่งข่าวสารไปยังอุปกรณ์

   หมายเหตุ : ถ้าไม่มีอุปกรณ์ต่อกับพอร์ต (PORT) T2 ก็จะเพิ่มจำนวนขึ้นด้วยเมื่อ GEIC พบสภาวะศูนย์จากทุกพอร์ต แต่เมื่อครบ 3 ครั้งแล้ว T2 ก็จะกลับเป็นศูนย์อีกครั้งหนึ่ง
3. T3 (REPLY TIMEOUT) เมื่อข่าวสารที่ต้องการตอบรับถูกส่งไปยังอุปกรณ์ GEIC จะจับเวลาและรอสัญญาณตอบรับจากอุปกรณ์ ถ้าอุปกรณ์ไม่ส่งสัญญาณตอบรับ GEIC ก็จะ TIMEOUT และ T3 จะนับเพิ่มขึ้น

ตัวอย่าง : GEIC ส่ง S1F2 ไปยังอุปกรณ์และคาดหวัง S1F2 สัญญาณ ตอบรับจากอุปกรณ์แต่อุปกรณ์ไม่ส่ง S1F2 ออกแล้ว GEIC จะ TIMEOUT รอ S1F2 ตอบ

4. T4 (INTERBLOK TIMEOUT) ถ้าอุปกรณ์กำลังมัลติบล๊อค (MULTIBLOCK) SECS MESSAGE ไปยัง GEIC และยังส่งข่าวสารไม่สมบูรณ์แล้ว GEIC จะ TIMEOUT และ T4 จะนับเพิ่มขึ้น

ตัวอย่าง : ถ้าอุปกรณ์ส่งข่าวสารที่ประกอบด้วย 5 บล๊อคนั้น หลังจากส่งข่าวสารออกไป 2 บล๊อคแล้วอุปกรณ์ยังไม่ส่งข่าวสารบล๊อคที่ 3,4 และ 5 ที่เหลือ GEIC จะ TIME OUT รอบล๊อคถัดไป

- BUFFER บนจอ PRO-DIAG คืออะไร

BUFFER แสดงจำนวน SECS MESSAGE บล๊อคที่ GEIC มีอยู่

- STATE คืออะไร

VSII มีการสื่อสารอยู่ 2 รูปแบบ คือ PXI ซึ่งใช้ติดต่อกับ HOST (ระบบ VAX) และ XPC ซึ่งเป็นการติดต่อแบบมัลติพอร์ต เพื่อที่จะให้ GEIC สามารถบริการอุปกรณ์พอร์ตทั้งหลายได้ในเวลาเดียวกัน เราจึงใช้ STATE เพื่อกำหนดสภาวะที่แตกต่างของแต่ละพอร์ต

STATE 0 = เป็นการแสดงภาวะว่าง (IDLE) ภายนอก ไม่มีอุปกรณ์ถูกเชื่อมโยงหรืออุปกรณ์ไม่มีการติดต่อสนทนา

STATE 1 = เป็นการแสดงภาวะว่าง (IDLE) ภายในสาย อุปกรณ์ถูกเชื่อมโยงเพื่อจัดสร้างข่ายสื่อสารกับ GEIC ในช่วงเวลาที่พอร์ตว่างไม่มีข่าวสารส่งออกหรือรับเข้านั้นในสภาวะนี้ GEIC สามารถรับ ENQ และส่ง ENQ กับอุปกรณ์ได้

ถ้าได้รับ ENQ จะเซ็ต (SET) STATE = 6

ถ้าส่ง ENQ จะเซ็ต STATE = 4

STATE 2 = เป็นสภาวะที่ GEIC จะรอเพื่อตรวจสอบความยาวไบต์ (LENGTH BYTE) ว่าถูกต้อง (10 ถึง 255)

ถ้าความยาวไบต์ถูกต้องจะเซ็ต STATE = 3

STATE 3 = ในสภาวะนี้จะได้รับจำนวนของไบต์ข่าวสารซึ่งระบุความยาวไบต์ที่ได้รับใน STATE 2 หลังจากไบต์ทั้งหมดรวมทั้งไบต์ตรวจสอบ (CHECKSUM) 2 ไบต์ด้วยแล้วก็จะทำการตรวจสอบ
ถ้าการตรวจสอบถูกต้องจะเซ็ต STATE = 8
ถ้าการตรวจสอบไม่ถูกต้องจะเซ็ต STATE = 7

STATE 4 = สภาวะนี้จะรอการตอบสนอง EOT จากอุปกรณ์หรือรับ ENQ จากอุปกรณ์
ถ้าได้รับ EOT จะเซ็ต STATE = 5
ถ้าได้รับ ENQ จะเซ็ต STATE = 6

STATE 5 = เป็นสภาวะที่เริ่มส่ง SECS MESSAGE ไปยังอุปกรณ์ หลังจากไบต์ทั้งหมดรวมทั้งไบต์ตรวจสอบ 2 ไบต์ ถูกส่งออกแล้วจะเซ็ต STATE = 9

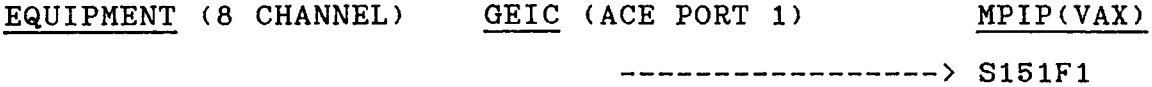
STATE 6 = เป็นสภาวะที่ตรวจสอบ SECS บัฟเฟอร์ (BUFFER) มีหรือไม่ ถ้ามีก็จะส่ง EOT ไปยังอุปกรณ์ และเซ็ต STATE = 2 ถ้าไม่มีจะเซ็ต STATE = 1

STATE 7 = เป็นสภาวะการเสร็จสิ้น ถ้ามีความผิดพลาดเกิดขึ้นในสภาวะใด ๆ ก็จะทำการขจัดทิ้ง CHARECTER ทั้งหมดที่ได้รับจากอุปกรณ์จนกระทั่งไม่มีไบต์ถูกส่งออกแล้วจึง TIMEOUT และส่ง NAK ไปยังอุปกรณ์แล้วกลับคืนสู่ STATE = 1

STATE 8 = เป็นสภาวะที่ส่ง ACK ไปยังอุปกรณ์ และผ่านข่าวสารที่สมบูรณ์ไปยัง MESSAGE RECEIVE แล้วจึงกลับสู่ STATE = 1

STATE 9 = เป็นสภาวะที่รอ ACK หรือ NAK จากอุปกรณ์เมื่อได้รับแล้วจะกลับคืนสู่ STATE = 1

- ลำดับขั้นตอนของการเริ่มต้น GEIC มีดังนี้
เมื่อเริ่มต้น GEIC จะยังเงียบ " QUIET " อยู่จนกระทั่ง MPIP สนทนา "TALK" กับมัน GEIC จึงเริ่มติดต่อกับอุปกรณ์



S152F17 ---------->

<---------- S152F18

MARK HOST ONLINE

BEGIN ZERO STATE POLLING

ON ALL EQUIPMENT PORT.

----------> S1F1 (ID = 0)

S9F1 <----------

----------> S1F1 (ID = EQUIPMENT ID)

S1F2 <----------

S151F167 <----------

ถ้าพอร์ตไม่ได้ถูกต่อเชื่อม ก็จะเกิดเป็นดังนี้คือ

----------> ENQ

TIMEOUT T2 = 1

----------> ENQ

TIMEOUT T2 = 2

----------> ENQ

TIMEOUT T2 = 3

CLEAR PORT BUFFER

T2 = 0

บทที่ 6

6.1 MPIP SOFTWARE

MULTI PORT INTERFACE PROCESS (MPIP)

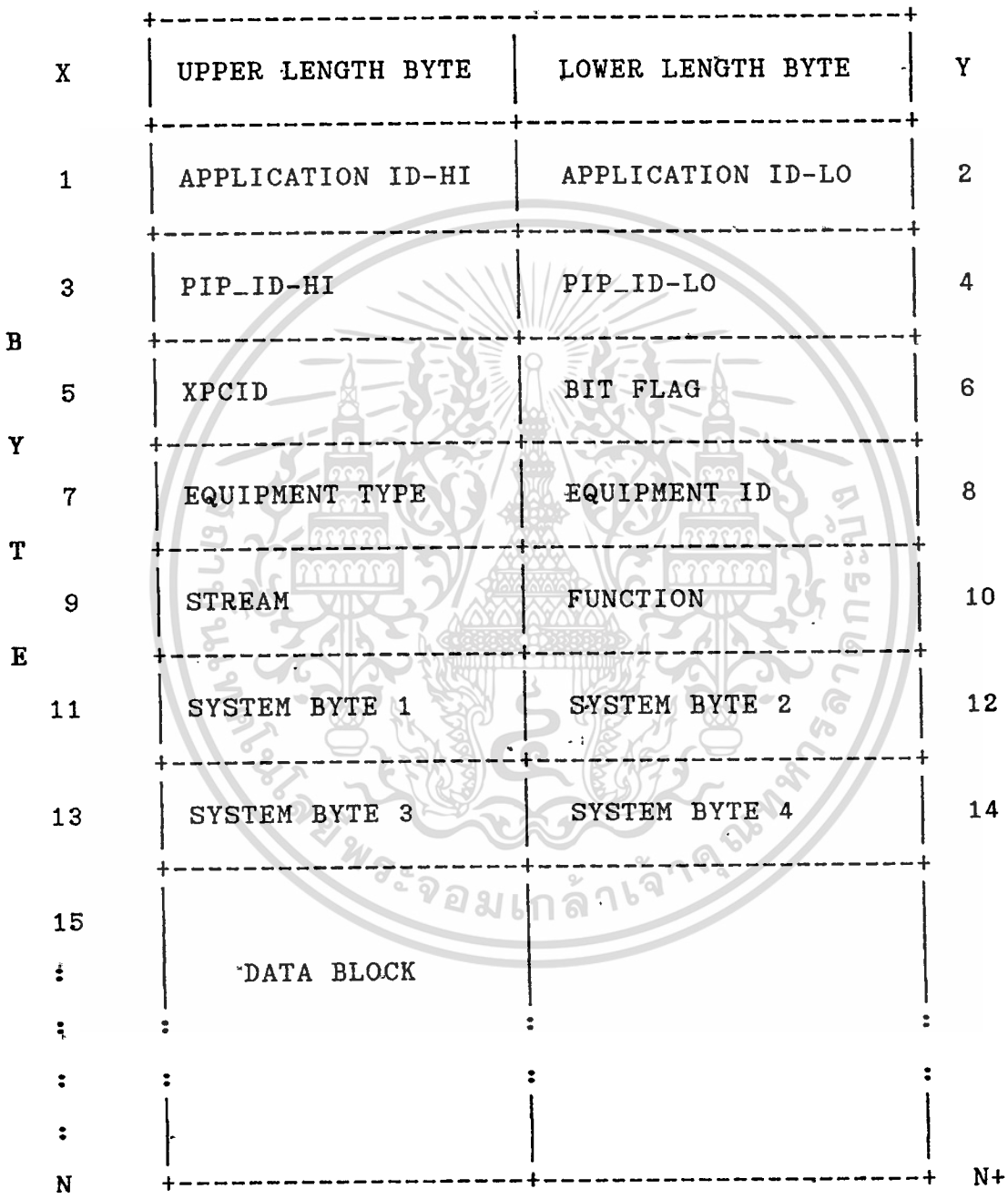
1.0 ลักษณะทั่วๆไป

MPIP (MULTI PORT INTERFACE PROCESS) เป็นโปรแกรมสื่อสารติดต่อซึ่งใช้ PXI (PIP_XPC_INTERFACE) โปรโตคอลติดต่อไปยัง GEIC (GENERIC EQUIP MENT INTERFACE COTROLLER) ผ่านทาง DEC SERVER POINTS.
โดยการใช้ MPIP ทำให้การใช้งานสามารถพัฒนาให้รับ/ส่ง ข่าวสาร PXI จาก หรือไปยังอุปกรณ์และทำฟังก์ชั่นต่าง ๆ เหมือน REMOTE CONTROL ของอุปกรณ์ สามารถที่จะ DOWNLOAD/UPLOAD โปรเซสโปรแกรม เก็บรวบรวมข้อมูลโดยตรง จากอุปกรณ์และสามารถตรวจดู (MONITOR) สภาวะของอุปกรณ์ได้จาก VAX TERMINAL การสื่อสารโดยโปรแกรมใช้งาน (APPLICATION) ทำได้โดยผ่าน MAILBOX โดยไม่จำเป็นต้องทราบตำแหน่งของอุปกรณ์หรือต้องทราบว่าต่อเชื่อมกับ GEIC ใด เพียงแต่ทราบชื่อกำหนด (ID) ของอุปกรณ์ก็พอ การสื่อสารระหว่าง GEIC และอุปกรณ์นั้นใช้มาตรฐานข่าวสาร SECS จึงไม่พบปัญหาเรื่องการเข้ากัน ไม่ได้ GEIC จะแปลข่าวสาร PXI ให้เป็นข่าวสาร SECS ก่อนที่จะส่งออกไปให้ อุปกรณ์

2.0 โครงสร้างข่าวสาร PXI

ข่าวสาร PXI ประกอบด้วย HEADER 14 ไบต์ โดยเปลี่ยนแปลงความยาวของตัว มัน โครงสร้างของ HEADER มีดังนี้คือ

6.2 โครงสร้างของข้อมูล



BIT FLAG - BIT 0 = DIRECTION - 0 = H->X 1 = X->H

BIT 1 = REPLY - 0 = NO REPLY 1 = REPLY

BIT 2 = DATA TYPE - 0 = NORMAL 1 = SPOOL

BIT 3 - BIT 7 RESERVED

การใช้ความยาว 2 ไบต์ นั้นหมายความว่าขนาดของข่าวสารได้ถึง 64 K ไบต์ โดยทั่วไปแล้วจะเพียงสำหรับความต้องการส่วนใหญ่

หมายเหตุ : STREAM จะมีจำนวนเต็มไบต์ เท่ากับ 8 บิต ไม่ใช่ 7 บิต สำหรับมาตรฐาน SECS ซึ่งเป็นความแตกต่างระหว่าง SECS และ PXI

PXI HANDSHAKING โปรโตคอล จะคล้ายกันกับ SECS โปรโตคอล ดังนี้คือ

ENQ - REQIEST THE LINE TO SEND

EOT - READY TO RECEIVE

ACK - CORRECT RECEPTION **

NAK - INCORRECT RECEPTION **

** FOR BOTH BLOCK AND END OF MESSAGE.

BLOCK ACKNOWLEDGEMENT = 50 BYTES/BLOCK FOR RECEIVE

TIMEOUT

T1 = 15 SECS - INTERCHARACTER TIMEOUT CUM FOR 50 BYTES

T2 = 8 SECS - PROTOCOL TIMEOUT FOR EOT,ACK

T3 = 60 SECS - REPLY TIMEOUT.

RTY = 3 - RETRY LIMIT

BAUD RATE = 9600, WITH 1 MS BETWEEN CHARACTER (FOR RECEIVING);

3.0 PREAENCE POLLING

โดยประมาณช่วง 1 นาที MPIP จะใช้พอร์ตตอบสนองโดยการส่ง ENQ เมื่อ MPIP ได้รับ EOT แล้วมันจะส่งค่า 0 ออกไป 2 ไบต์ (NULL) โดย XPC จะตอบสนองด้วย ACK เมื่อลำดับขั้นตอนเสร็จสิ้น MPIP จะพิจารณาว่าพอร์ตนั้นยังทำงาน ถ้าเวลาใดที่ลำดับขั้นตอนผิดพลาดแล้ว MPIP จะพิจารณาว่าพอร์ตถูกตัดขาดหรือล้มเหลวและพยายามให้มันเริ่มทำงานอีก

4.0 PORT STARTUP

เมื่อ MPIP เริ่มทำงานครั้งแรกมันจะพยายามสนทนากับพอร์ตโดยการส่ง ENQ ไป หลังจากพยายาม 3 ครั้งแล้ว ถ้าไม่ได้ตอบสนองมันจะถือว่า XPC OFFLINE และ

งาน (STARTUP) อีก

ถ้าได้รับ EOT มันจะส่งข่าวสาร STARTUP ไปยังพอร์ต,STREAM 151 (S151) FUNCTION 1 (F1) GEIC จะตอบสนองด้วย S151 F2 พร้อมทั้ง ID ของมันในข่าวสาร HEADER เมื่อแล้วเสร็จพอร์ตจะถูกพิจารณาว่ายังทำงาน และจะ STARTUP อย่างไรก็ดี ถ้าได้รับ EOQ แล้ว MPIP จะตอบสนองด้วย EOT เพื่อให้ GEIC สนทนากับมัน

5.0 CUSTOMIZING MPIP สำหรับอุปกรณ์บางชนิด

มีอยู่ 2 โมดูลซึ่งต้องการดัดแปลงเพื่อที่จะ CUSTOMIZE MPIP สำหรับอุปกรณ์บางชนิดโดยเฉพาะ คือ

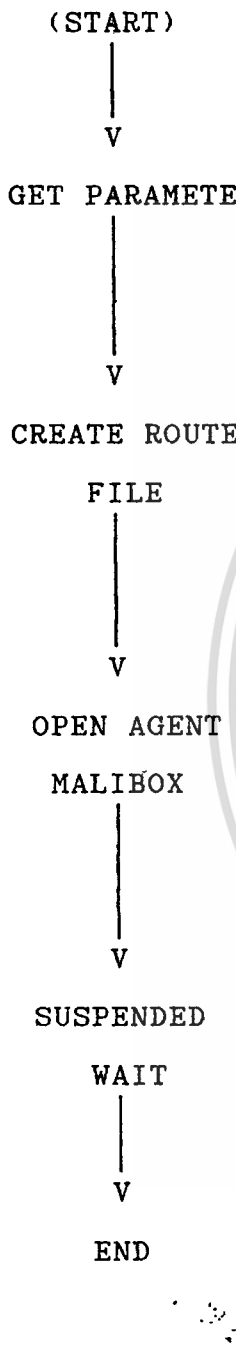
5.1 MPIP__CREATE_AGENT_MAILBOX จะแยกออกจากโมดูลหลักในโมดูลนี้ต้องการ MAILBOX ในตอนเริ่มต้นโดยปกติจะมีชื่ออยู่แล้วเช่น HUNKAR_MBOX เป็นต้น

5.2 MPIP__PROCESS_AGENT_MESSAGE เป็นโมดูลซึ่งแยกออกจากโมดูล PROCESS_PORT_MESSAGE ในโมดูลนี้ข่าวสาร STREAM และฟังก์ชั่นก็จะถูกกำกับใหม่ให้เหมาะสม AGENT สำหรับ PROCESSING ในทุกแห่งที่ข่าวสารเฉพาะเจาะจงถูกส่งออกไปนั้น ตัวแปร MESSAGE_PROCESSED ซึ่งนิยามไว้ PROCESS_PORT_MESSAGE จะต้องถูกเซ็ตไป "จริง" (TRUE)

STARTUP FLOW CHART

PHASE 1 : INITIALIZATION

ในเฟส (PHASE) แรกของการรัน MPIP ประกอบด้วยการเริ่มต้นของตัวแปร GLOBAL ต่าง ๆ อ่าน ARGUMENT จากคำสั่งรัน,สร้างไฟล์ ROUTE TABLE, เปิด MALIBOX สำหรับตัวเองและของ AGENTS ท้ายที่สุด MPIP จะจัดแจงหมายเลขช่อง (CHANNEL) ให้กับพอร์ตและทำงานทั้งหลายและไปยังสภาวะนิ่งสงบ



OBTAIN PARAMETERS ON EQUIPMENT TYPE, NUMBER OF PORTS TO SUPPORT, AND LOGICAL NAMES OF PORTS TO BE SUPPORTED.

CREATE ROUTE TABLE FILE TO CONTAIN ID OF XPC, EQ_ID AND AN INDEX NUMBER.

ASSIGN CHANNEL NUMBERS (APPL_ID)FOR MAILBOX FOR AGENTS, SO AS TO FORWARD SPECIFIC MESSAGE TO THEM.

DEFINES/ELABORATE TASK.

ASSILGN CHANNEL NUMBERS TO SUPPORTED PORTS.

ACTIVATE TASKS TO RECEIVE DATA FROM PORTS.

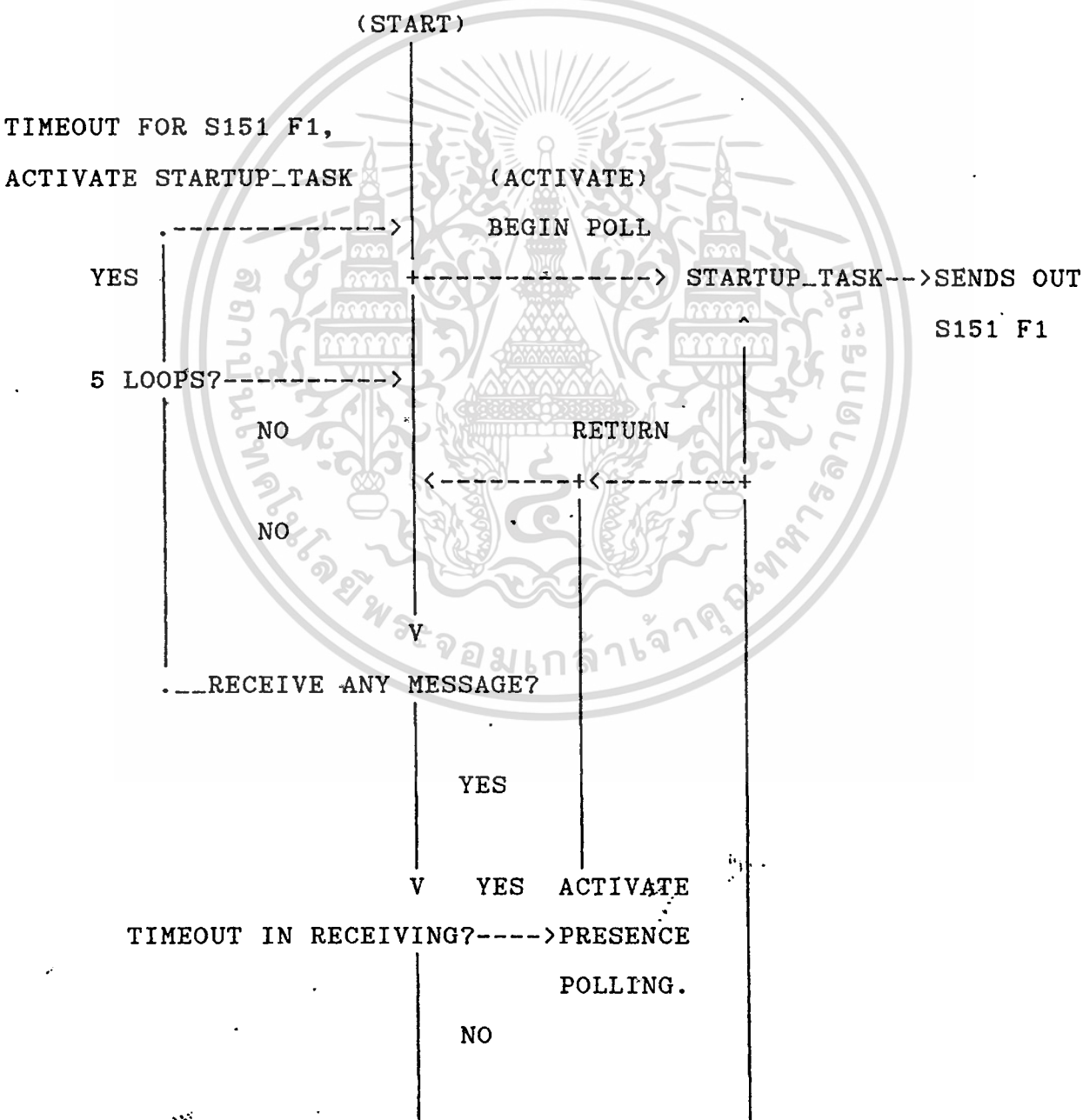
GO INTO A HIBERNATE STATE.

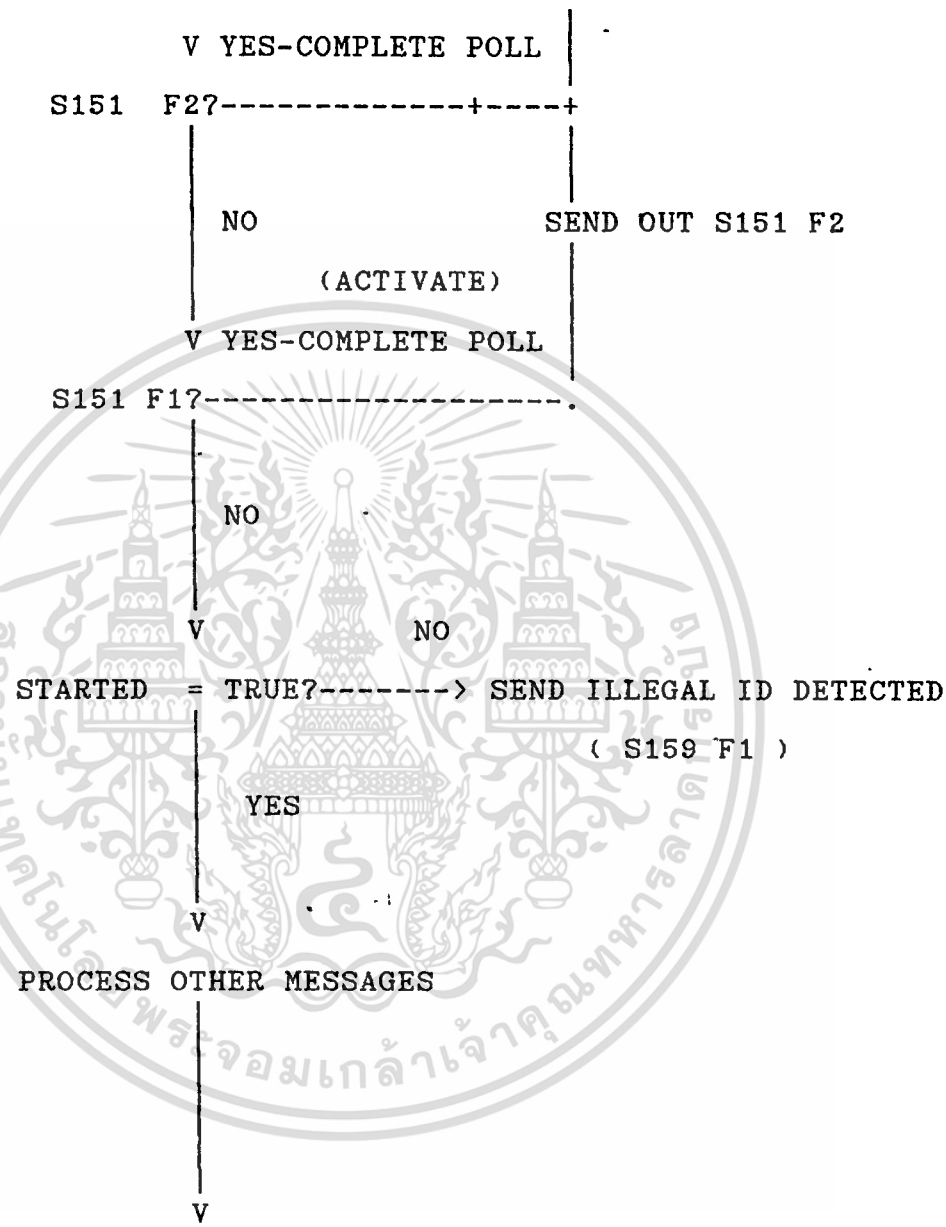
PHASE 2 : STARTUP OF PORTS

PORT_IN_TASK เป็นกุญแจงานหลักใน MPIP มันจะทำให้ภาระต่าง ๆ ทำงานคือ START UP_TASK,PORT_OUT_TASK และโดยทางอ้อม PRESENCE_TASK ก่อนที่จะทำการเข้าไปยังพอร์ตได้จะต้องขอไปยัง PORT_MANAGER_TASK เพื่อได้สิทธิเข้ายังพอร์ต หลังจากการส่งหรือรับผู้เรียก (CALLER)จะต้องปล่อยสิทธินั้นเสียก่อนเพื่อให้วิธีการอื่น ๆ ในโปรเซสสามารถเข้าไปหามันได้ ในทุก ๆ LOOP PORT_IN_TASK จะสงบนิ่งประมาณ 2 วินาทีอยู่ที่พอร์ต ถ้า ENQ ถูกส่งโดย XPC มิเช่นนั้นก็จะออกจาก LOOP ด้วยสภาวะ NO_DATA

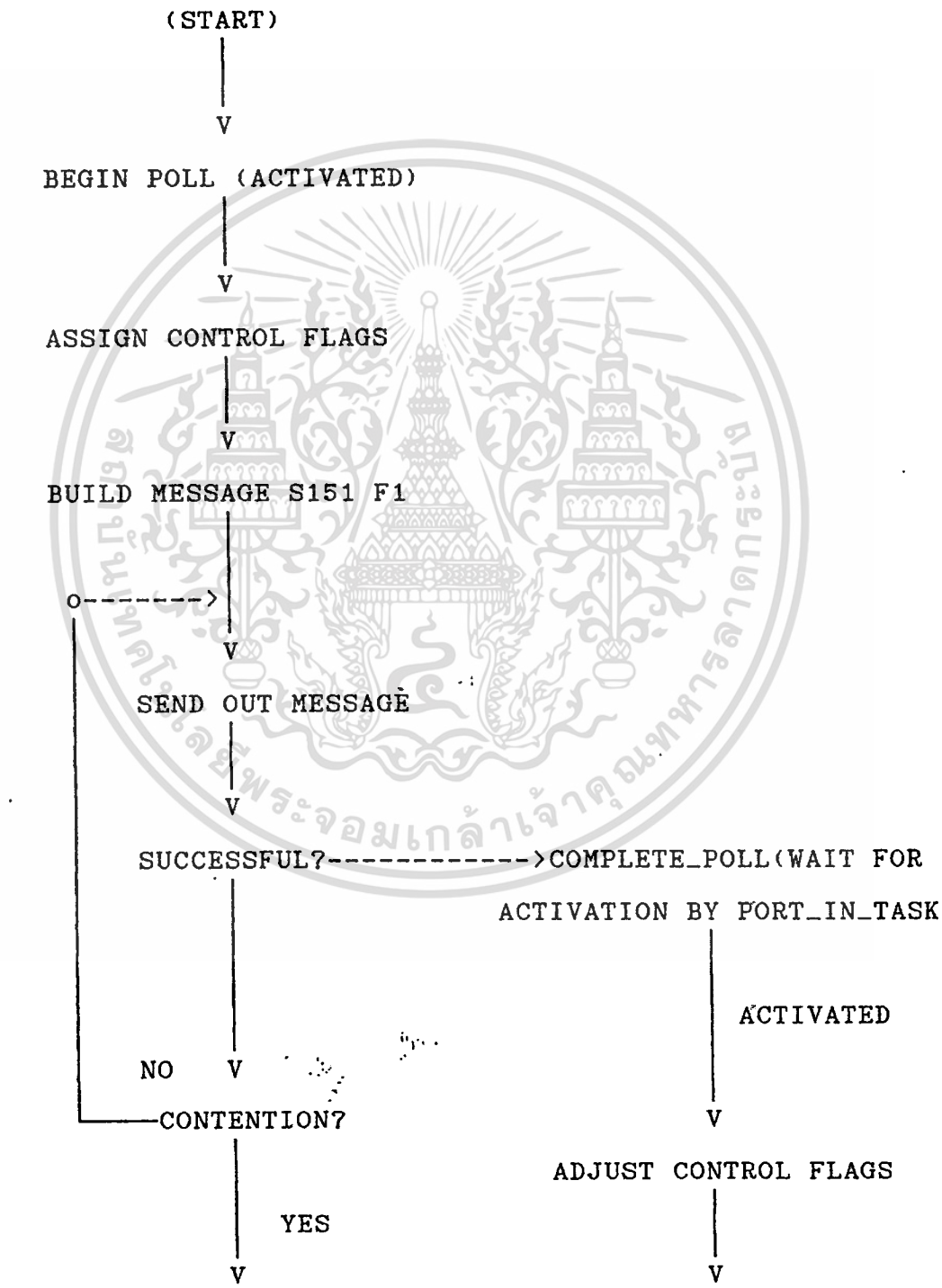
THE DIAGRAM BELOW IS A SIMPLIFIED PROGRAM FLOW FOR THE TASK PORT_IN_TASK.

PORT_IN_TASK ALGORITHM (ONE TASK FOR EACH PORT)




V YES-COMPLETE POLL
S151 F2?
NO
SEND OUT S151 F2
(ACTIVATE)
V YES-COMPLETE POLL
S151 F1?
NO
V
NO
STARTED = TRUE?
SEND ILLEGAL ID DETECTED
(S159 F1)
YES
V
PROCESS OTHER MESSAGES
V
LOOP BACK TO RECEIVE MESSAGES

STARTUP_TASK ALGORITHM



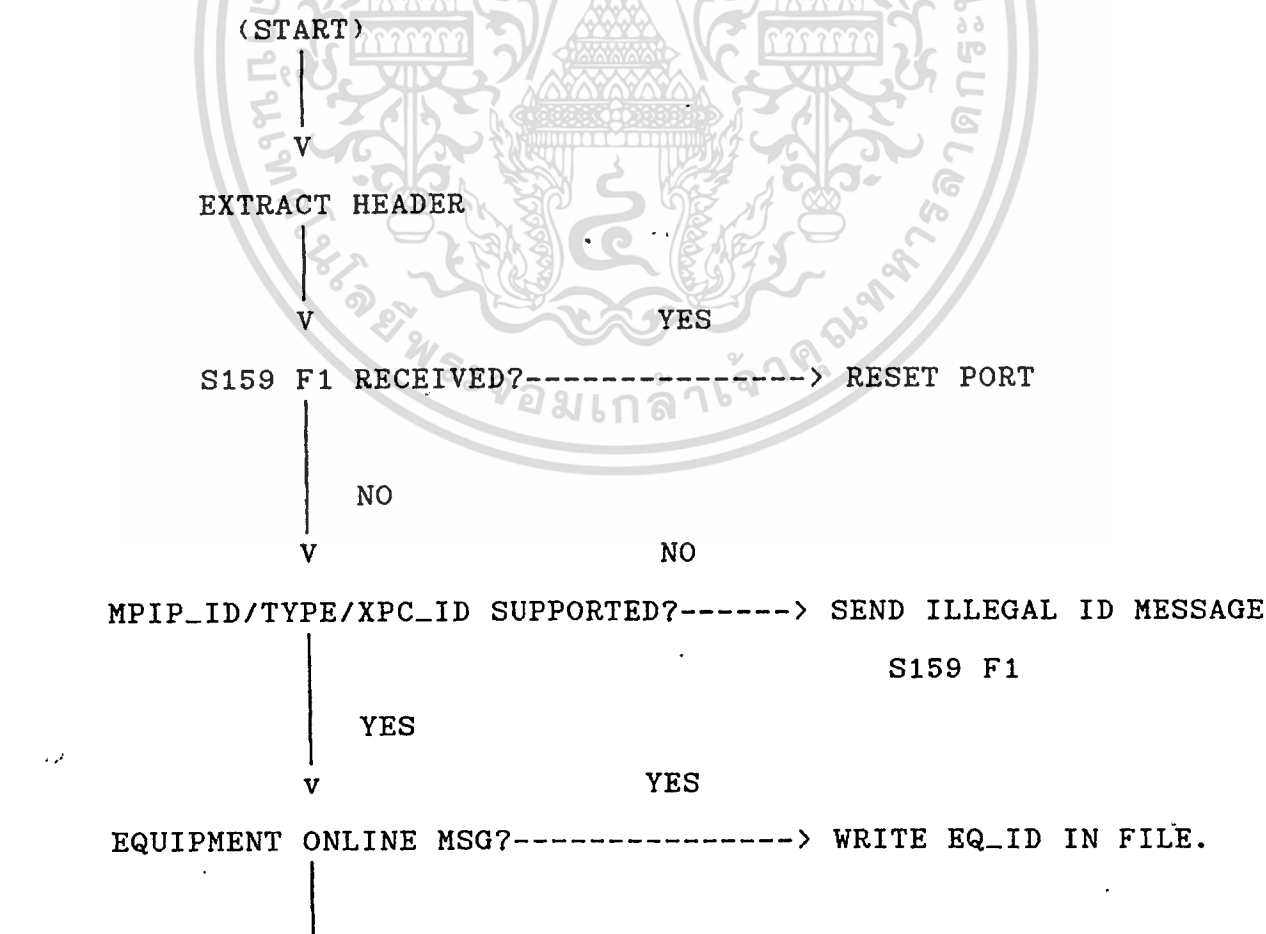
WE LET PORT_IN_TASK.

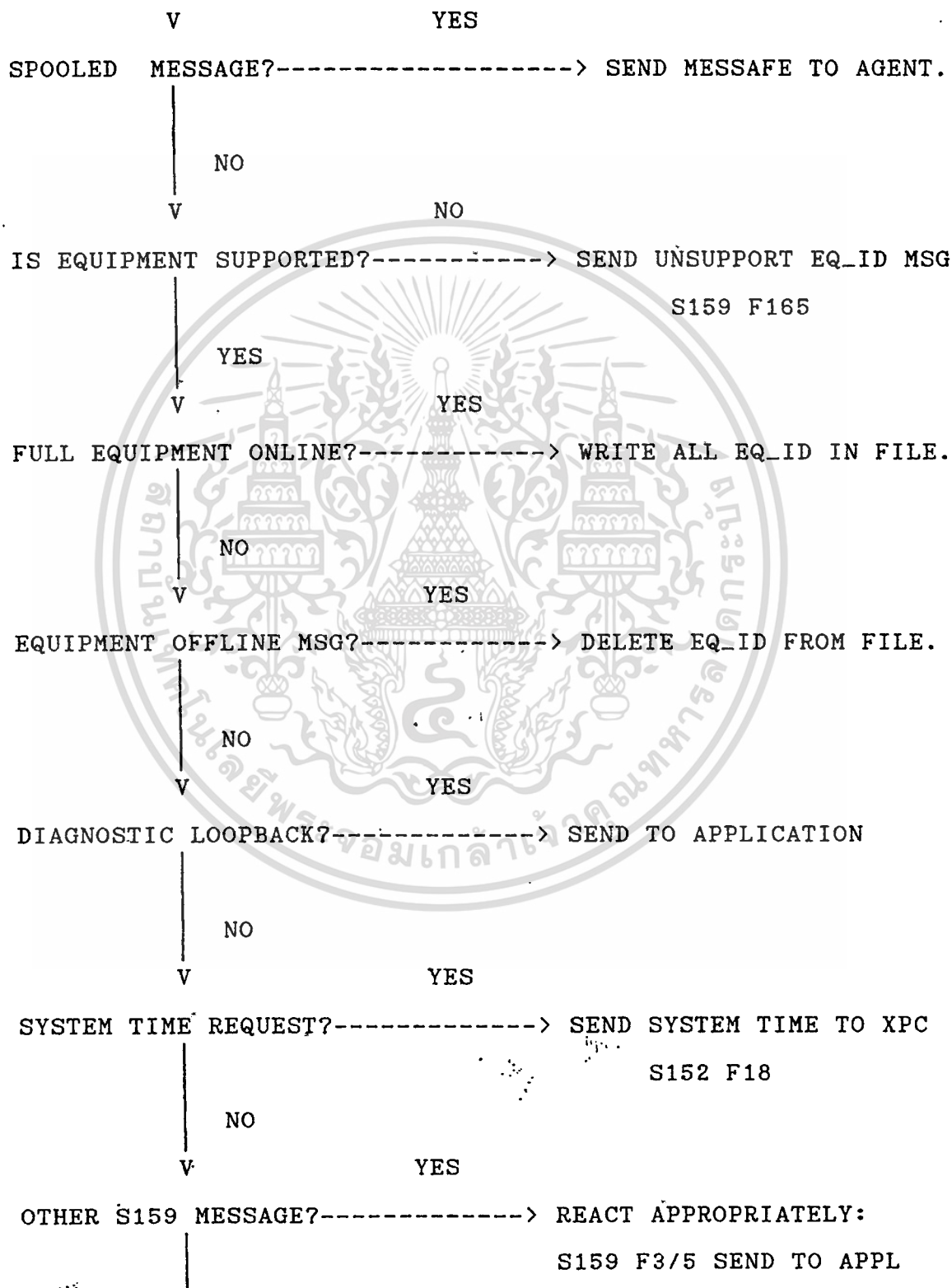
TO TRIGGER US AGAIN.

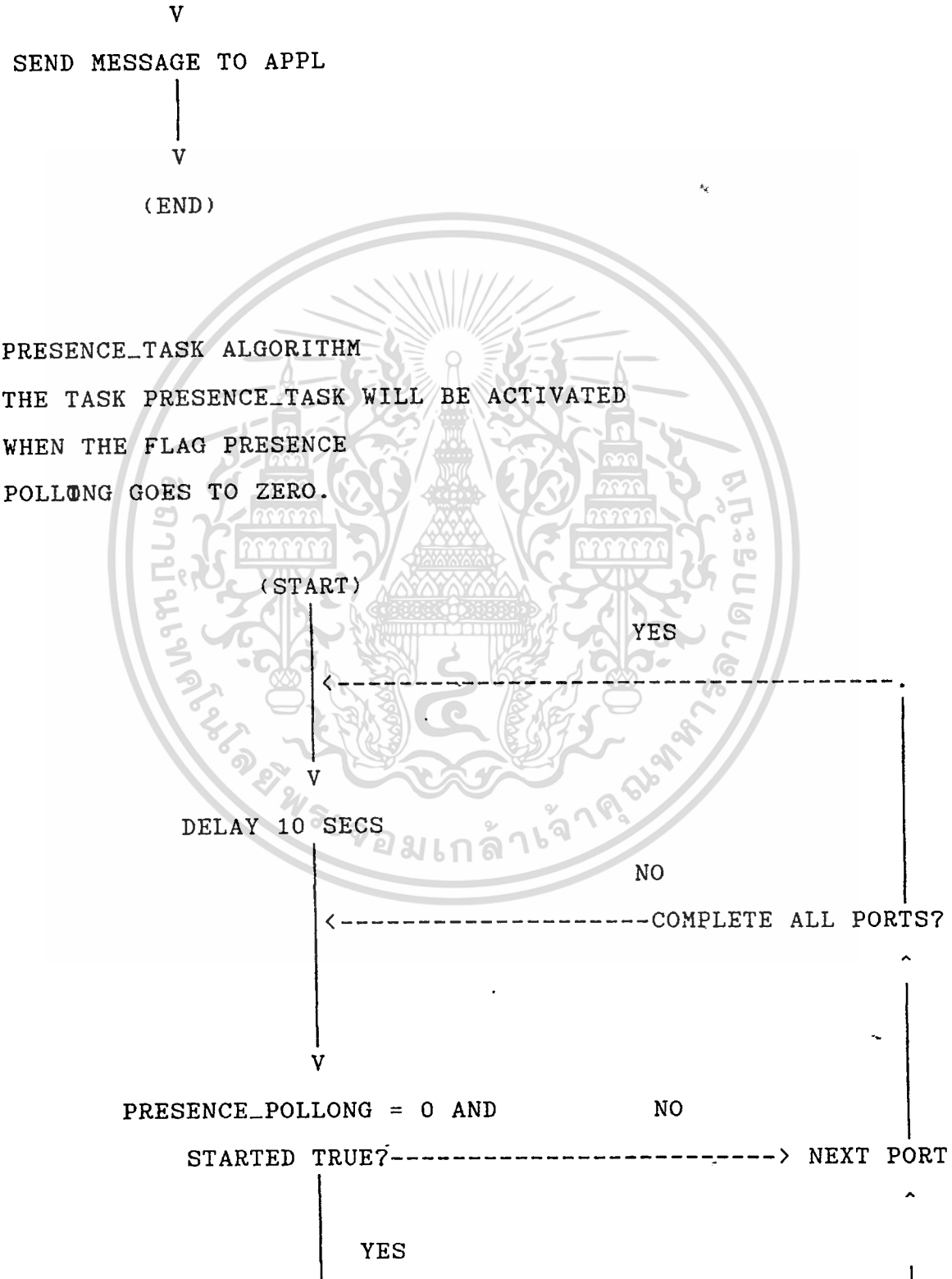
PHASE 3 : WAITING GOR MESSAGES.

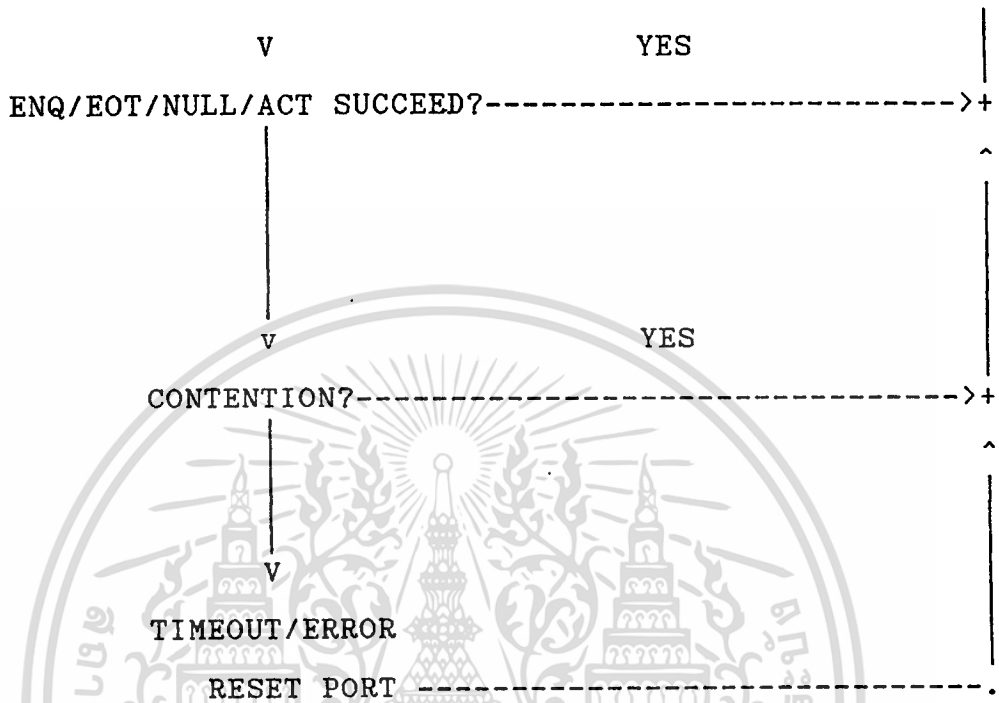
ในทันทีพอร์ตเริ่มทำงาน STARTUP_TASK ก็จะหมดลงไป PORT_IN_TASK จะไปวน LOOP เพื่อคอยที่พอร์ตอยู่ 2 วินาที (ซึ่งจะอยู่ในสภาวะสงบนิ่ง) และกลับถ้าไม่ได้รับอะไร หลังจากวน LOOP ได้ 30 ครั้งโดยไม่มีสภาวะของข่าวสาร FLAG PRESENCE_POLLING จะถูกเซ็ตไปยัง 0 และไปทำให้ PRESENCE_TASK ทำงาน ถ้าไม่มีข่าวสารโปรโตคอลได้รับโดย PORT_IN_TASK,PROCESS_PORT_MESSAGE จะถูกเรียกตาม FLOW DIAGRAM ข้างล่าง

PROCESS_PORT_MESSAGE ALGORITHM



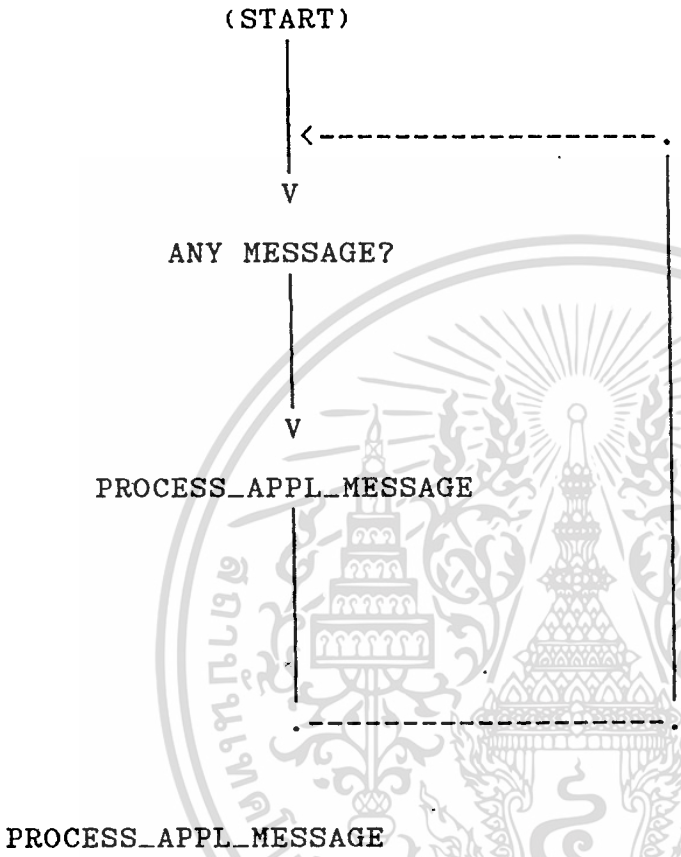

V
YES
SPOOLED MESSAGE?
SEND MESSAFE TO AGENT.
NO
V
NO
IS EQUIPMENT SUPPORTED?
SEND UNSUPPORT EQ_ID MSG
S159 F165
YES
V
YES
FULL EQUIPMENT ONLINE?
WRITE ALL EQ_ID IN FILE.
NO
V
YES
EQUIPMENT OFFLINE MSG?
DELETE EQ_ID FROM FILE.
NO
V
YES
DIAGNOSTIC LOOPBACK?
SEND TO APPLICATION
NO
V
YES
SYSTEM TIME REQUEST?
SEND SYSTEM TIME TO XPC
S152 F18
NO
V
YES
OTHER S159 MESSAGE?
REACT APPROPRIATELY:
S159 F3/5 SEND TO APPL


V
SEND MESSAGE TO APPL
V
(END)
PRESENCE_TASK ALGORITHM
THE TASK PRESENCE_TASK WILL BE ACTIVATED
WHEN THE FLAG PRESENCE
POLLONG GOES TO ZERO.
(START)
YES
V
DELAY 10 SECS
NO
COMPLETE ALL PORTS?
V
PRESENCE_POLLONG = 0 AND
STARTED TRUE?
NO
NEXT PORT
YES



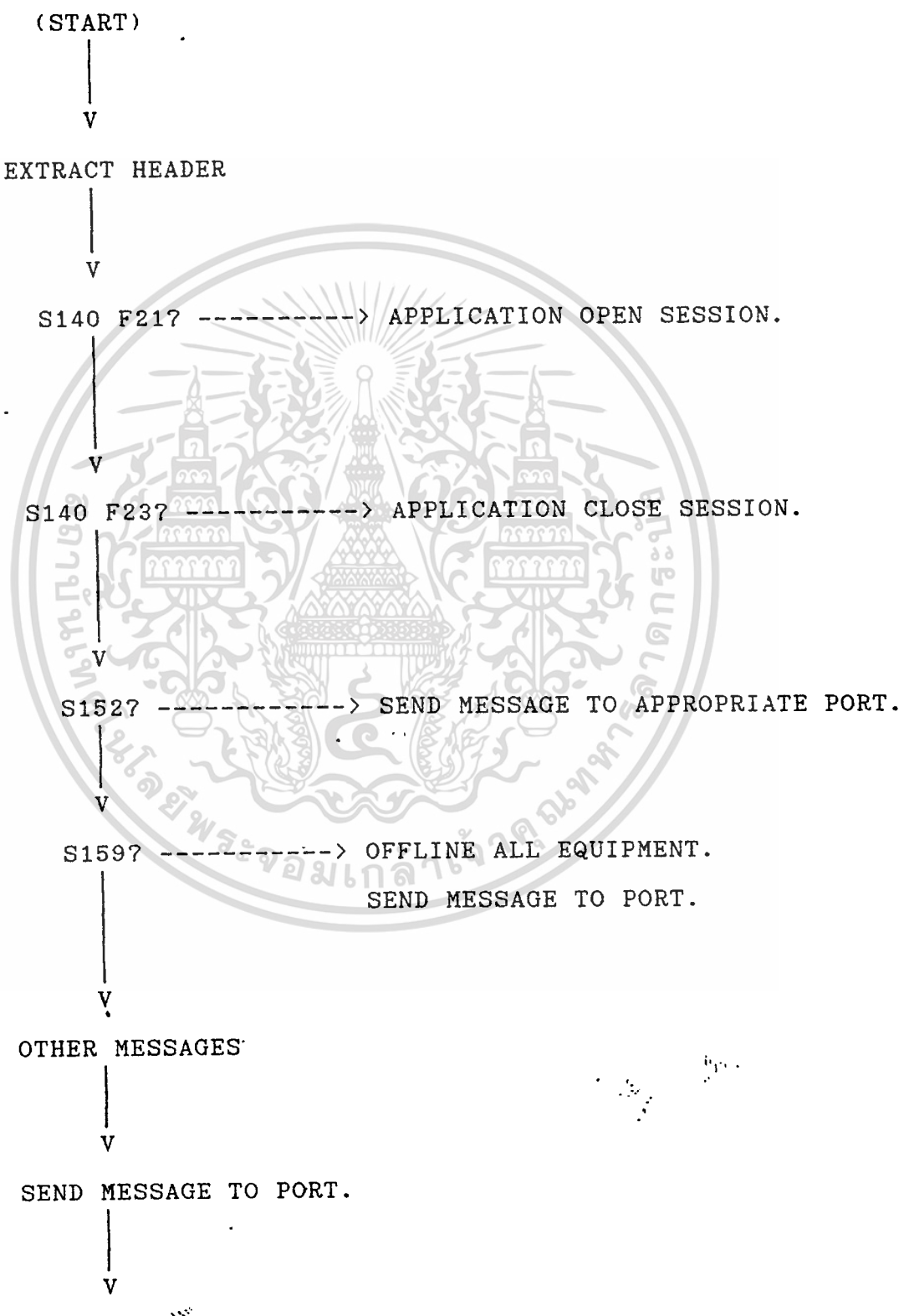
MBOX_IN_TASK ALGORITHM

MBOX_IN_TASK READS THE MESSAGE FROM ITS OWN MAILBOX PCMXX.SOME MESSAGE ARE FROM PROCESS_PORT_MESSAGE,TO BE FORWRDED TO THE PORT.



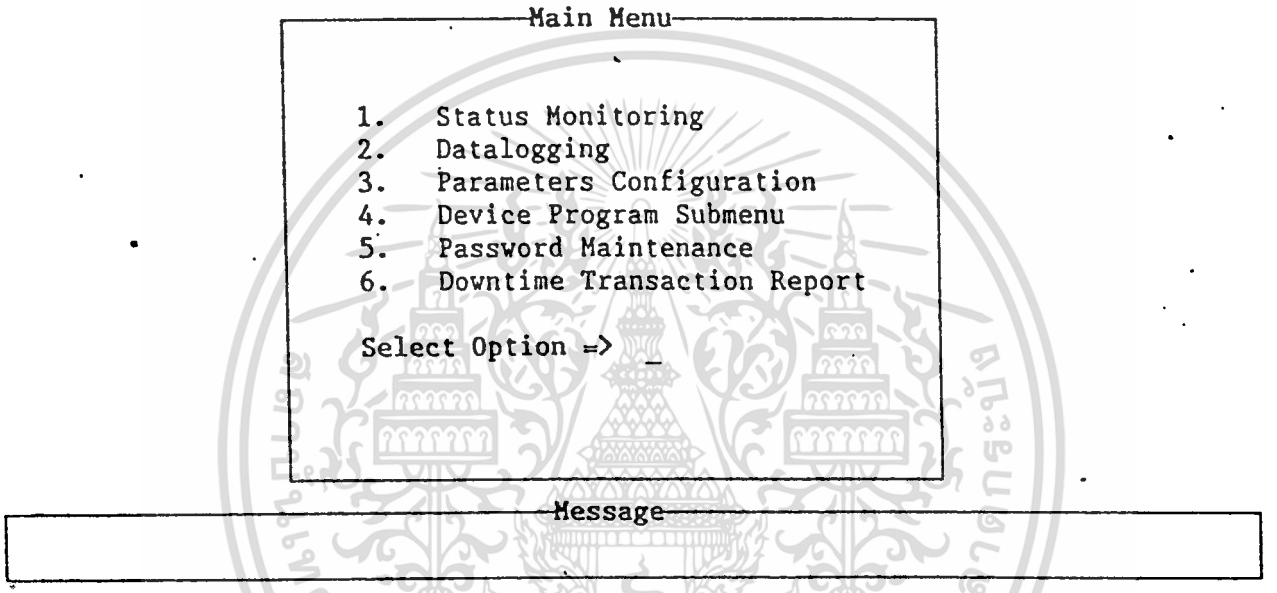
PROCESS_APPL_MESSAGE

THIS PROCEDURE PROCESSES THE MESSAGE RECEIVED BY MBOX_IN_TASK, AND ACTIVATES THE APPROPRIATE PORT_OUT_TASK TO FORWARD MESSAGES TO THE CORRECT PORT.


(START)
EXTRACT HEADER
S140 F21? ----------> APPLICATION OPEN SESSION.
S140 F23? ----------> APPLICATION CLOSE SESSION.
S152? ----------> SEND MESSAGE TO APPROPRIATE PORT.
S159? ----------> OFFLINE ALL EQUIPMENT.
SEND MESSAGE TO PORT.
OTHER MESSAGES
SEND MESSAGE TO PORT.

บทที่ 7

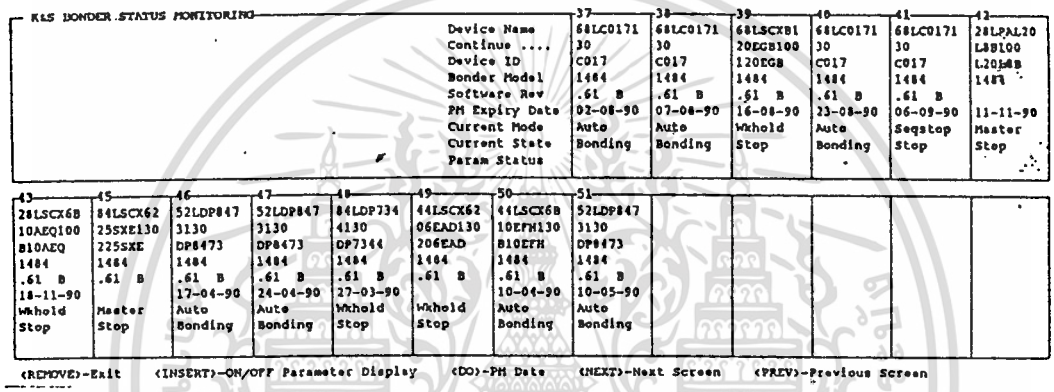
การใช้งานโปรแกรม



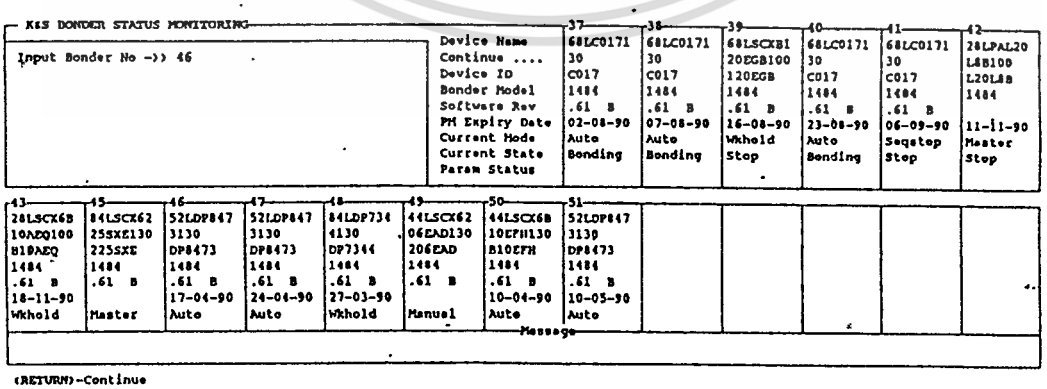
ผู้ใช้สามารถเลือก หมายเลขที่ต้องการทำงานแต่ละอย่างได้โดยมีทั้งหมด 6 ข้อ สำหรับ USER เลือกทำงาน

1. การรายงาน สถานะต่างๆ
2. การดู สภาพการทำงาน
3. การรายงาน โปรแกรม ของเครื่อง
4. การรายงาน หมายเลขของ DEVICE
5. การจัดการเกี่ยวกับระหัสผ่าน
6. รายงานการหยุดทำงาน

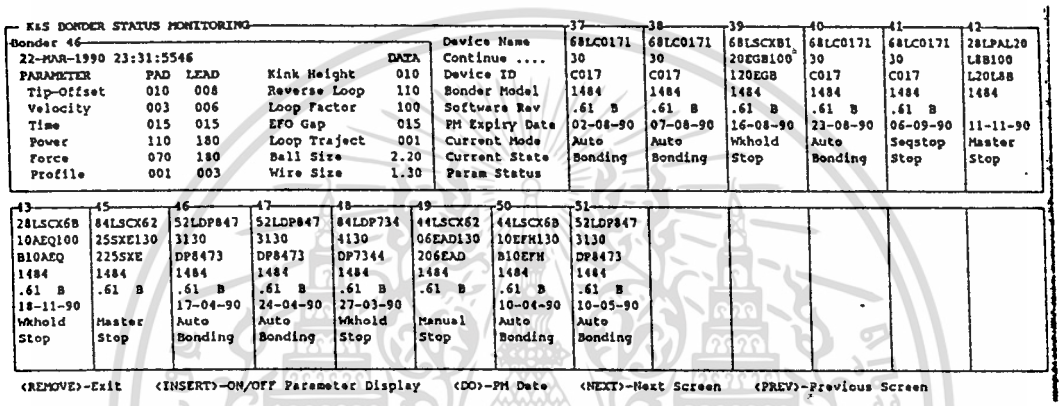
การเลือกข้อ 1 จะได้ข้อมูลจอภาพเป็นการรายงาน สถานะ และสามารถดูได้ครั้งละ 19 เครื่อง การเลือกรายงานครั้งละมากกว่า สามารถ เดินหน้า และ ย้อน หลัง โดยการกด KEY NEXT SCREEN และ KEY PREV SCREEN



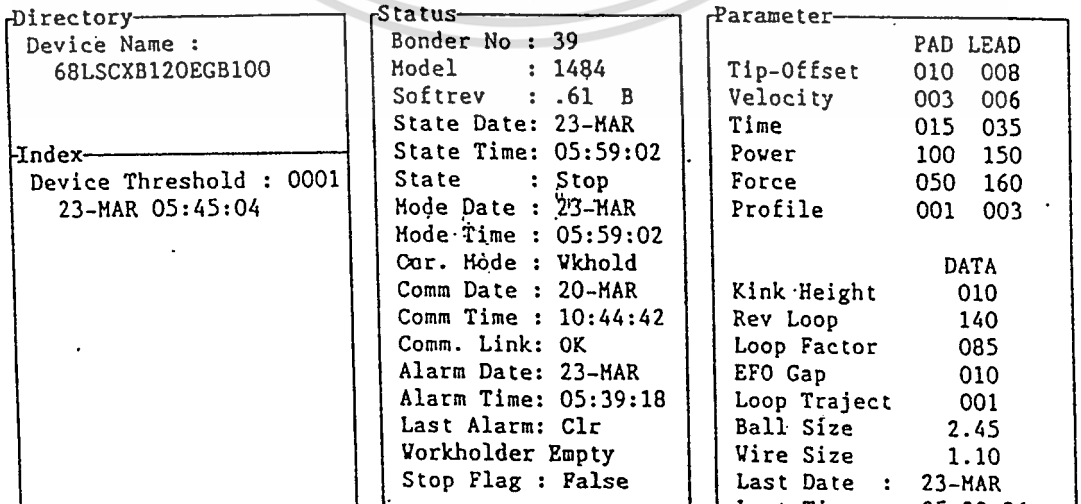
กรณีต้องการดูรายละเอียดต่อไปจะสามารถกด (INSERT) KEY สำหรับดูรายละเอียดแต่ละ M/C



การ SHOW PARAMETER จะสามารถใช้ KEY (UP) และ (DOWN) ในการดูรายละเอียด



การเลือก เมนู ข้อ 2 จะดูข้อมูลเกี่ยวกับตัวแปรสำหรับการทำงานของเครื่องในขณะนั้น โยจะแสดงข้อมูล PARAMETER ต่าง ๆ ของเครื่องและสามารถกด NEXT SCREEN หรือ PREV SCREEN สำหรับดูข้อมูลของแต่ละเครื่องได้



การเลือกข้อ 3 การาจัดการกำหนดค่า สูงสุดต่ำสุดของค่า PARAMETER ของเครื่อ โดยมี KEY ทำงานดังนี้

1. (FIND) สำหรับเลื่อกชนิด LEAD TYPE
2. (NEXT SCREEN) PARAMETER ตัวถัดไป
3. (INSERT) ใส่ PARAMETER ใหม่
4. (SELECT) ลบข้อมูล
5. (DO) SAVE ข็อมูล
6. (UP),(DOWN),(RIGHT),(LIFT) สำหรับเลือกแต่ละ PARAMETER

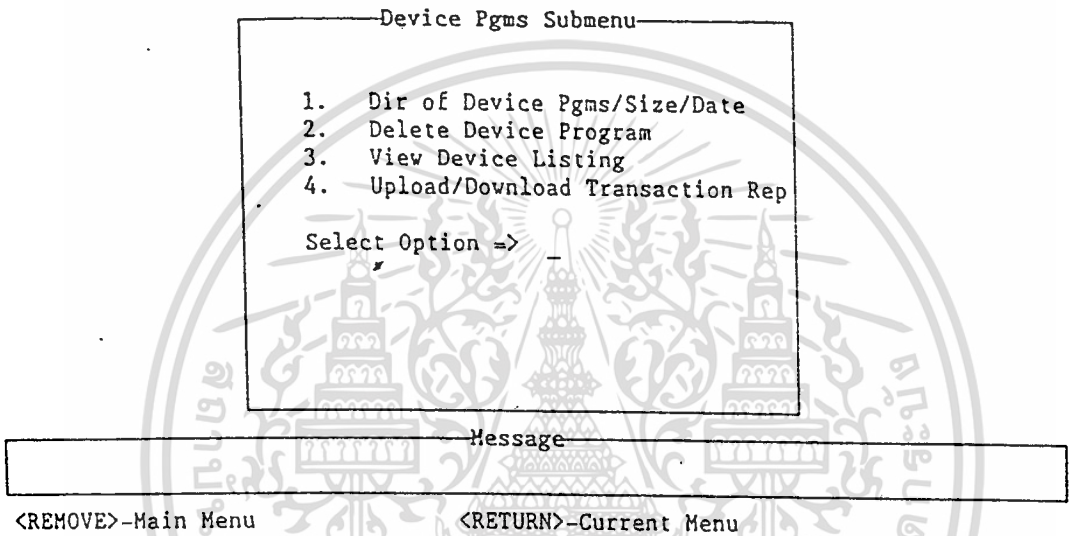
Lead & Wire Size : 68_100

Parameter	Pad		Lead	
	Lover	Upper	Lover	Upper
1. Tip-Offset	010	010	008	008
2. Velocity	003	003	006	006
3. Time	015	015	015	035
4. Power	100	120	105	160
5. Force	050	070	140	180
6. Profile	001	001	003	003
7. Kink Height	008	020		
8. Rev-Loop	080	150		
9. Loop Factor	080	100		
10. EFO Gap	010	030		
11. Loop Traj.	001	001		
12. Ball Size	220	260		

Message

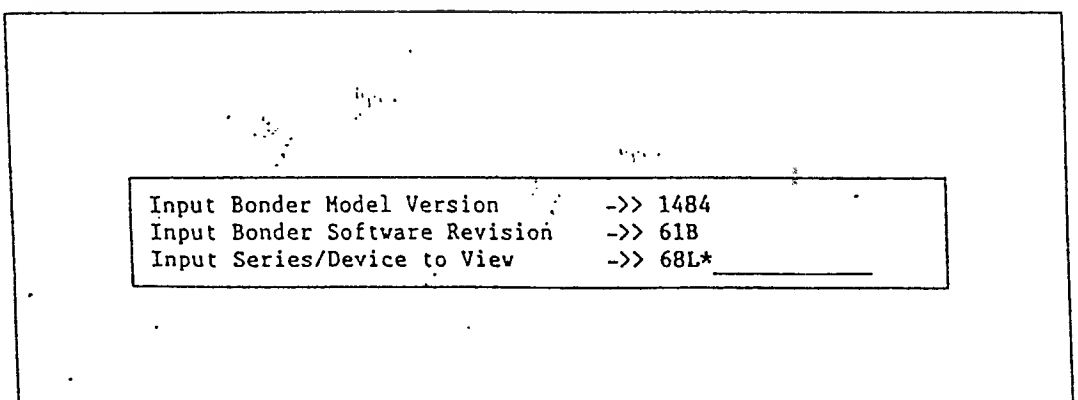
<REMOVE>-Exit <FIND>-Find Cfg <NEXT>-Next <INSERT>-Add <SEL>-Del <DO>-Save

การเลือกข้อ 4 การดูข้อมูล PROGRAM โดยแยกแต่ละ MENU ดังนี้



1. คือการ DIRECTORY ตามที่โปรแกรม , SIZE, วันที่
2. ลบโปรแกรมออก
3. ดู DEVICE ที่มีทั้งหมด
4. รายงานข้อมูลการ UP LOAD / DOWN LOAD

การกดข้อ 1 จะมีคำถามดังนี้



เมื่อเลือกได้ จะมีการ SHOW หน้าดังนี้

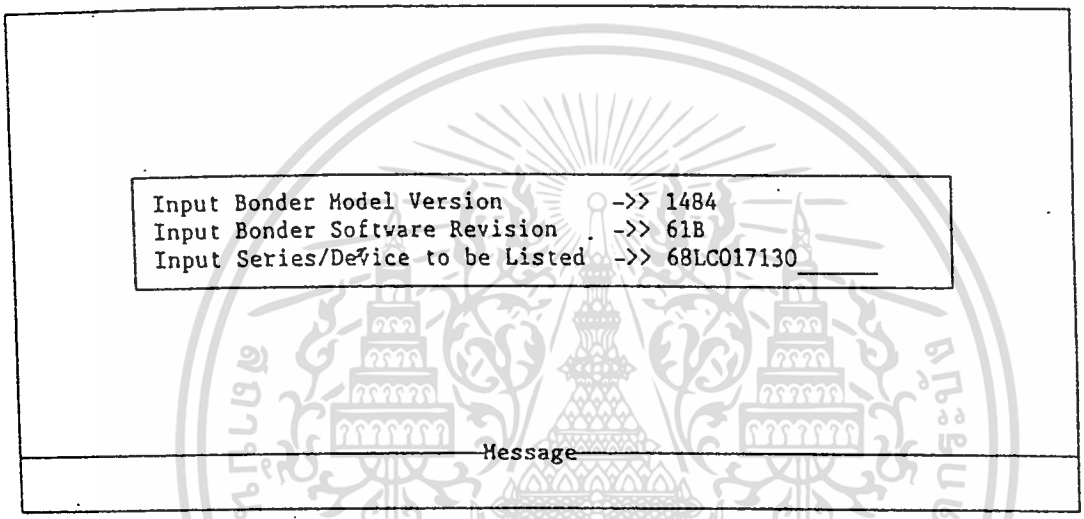
```
          Device Program                Size     Date      Time
          --------------                ----     ----      ----
          68LC017.DEV;1                   38   28-FEB-1990 08:57
          68LC017130.DEV;2                38   22-MAR-1990 21:08
          68LC11011130.DEV;1              28   15-MAR-1990 04:32
          68LCC1150V4130.DEV;1            40   14-MAR-1990 21:30
          68LDC017.DEV;2                  38   12-MAR-1990 03:52
          68LDC017130.DEV;2               38   12-MAR-1990 04:22
          68LDHPC46083130.DEV;1           40   13-MAR-1990 05:00
          68LDP8390130.DEV;1              34   12-MAR-1990 00:02
          68LDP8390C.DEV;2                34    1-MAR-1990 10:33
          68LDP8390C130.DEV;4             34   11-MAR-1990 15:53
          68LDP8421A.DEV;3                40    6-MAR-1990 22:33
          68LDP8421A150.DEV;1             40    8-MAR-1990 20:11
          68LDP8428.DEV;1                 34    2-MAR-1990 11:48
                                  Message
Press <RETURN> for More ........

<RETURN>-Exit    <REMOVE>-Break
```

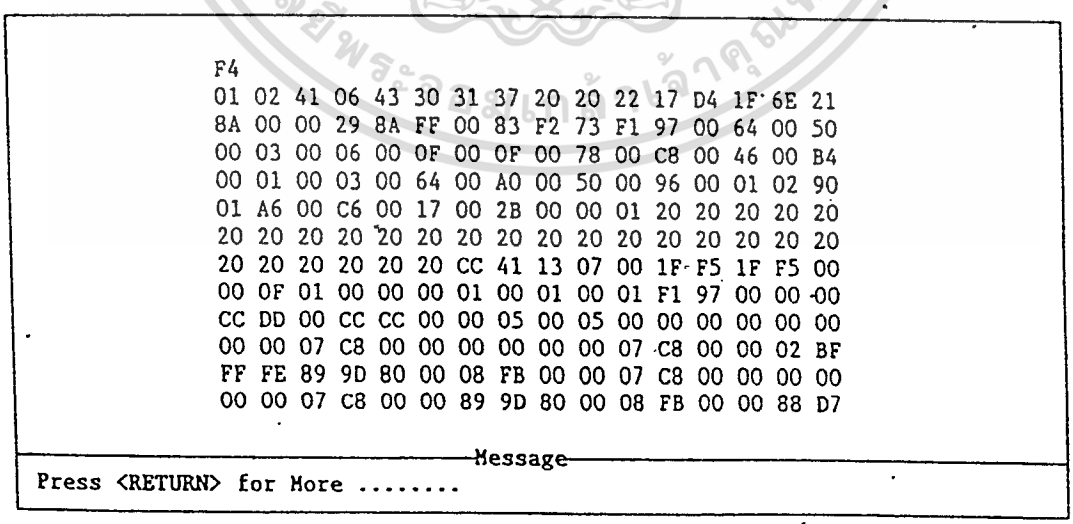
การกดข้อ 2 จะเป็นการDELETE PROGRAM โดยมีหน้าจะดังนี้

```
Input Bonder Model Version          ->> 1484
Input Bonder Software Revision      ->> 61B
Input Series/Device to be Deleted -->> 68LC017130_____
```

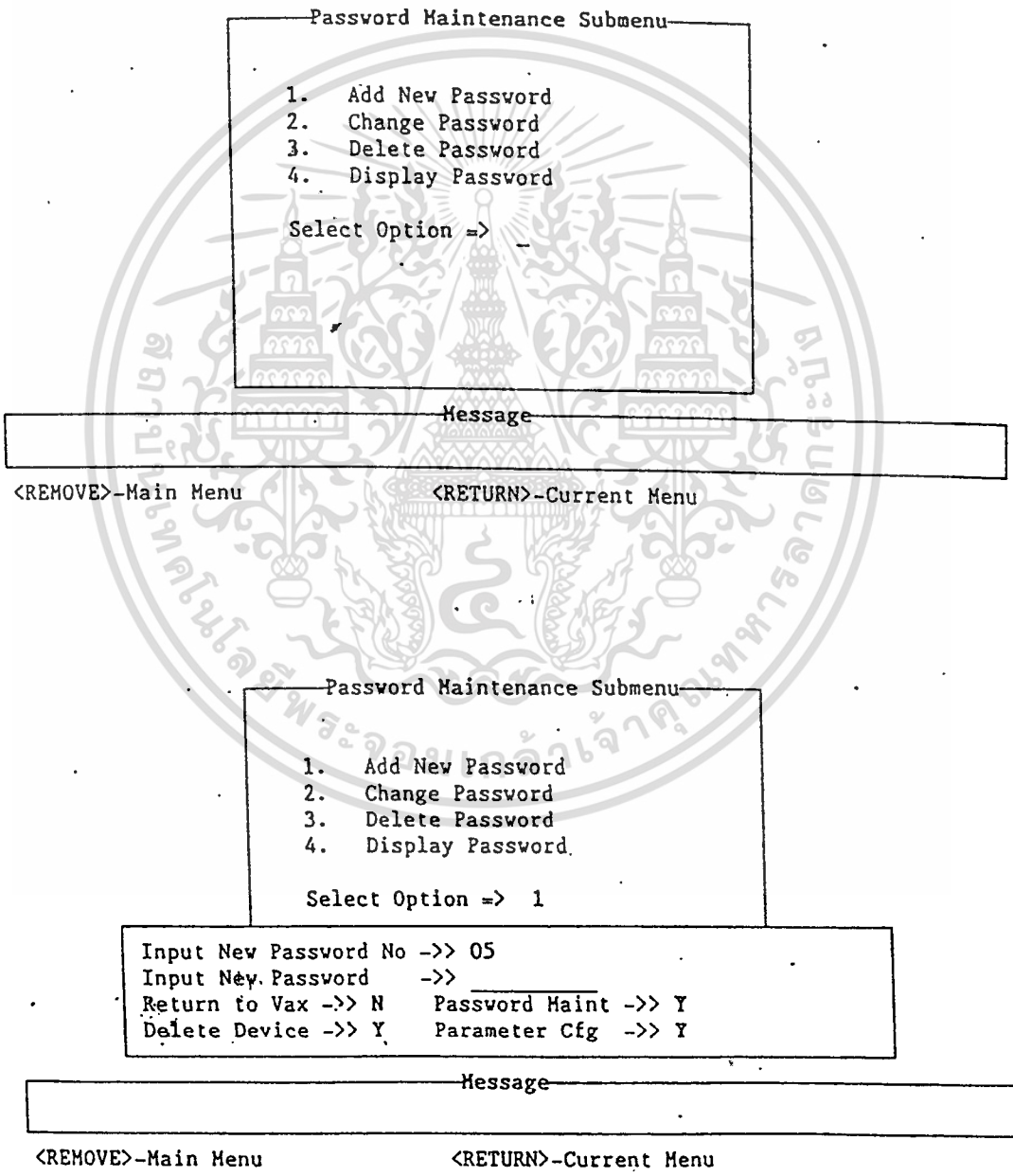
```
                                  Message

<RETURN>-Exit
```

การกดข้อ 3 จะเป็นการ ดูข้อมูลในแต่ละ DEVICE โดยการตอบคำถามดังนี้





การเลือกข้อ 5 จะเป็นการเปลี่ยนแปลง PASSWORD



การเลือกข้อ 6 จะเป็นการ REPORT DOWNTIME

```
          ID  Device Program     Act Date          Time
          --  --------------     --- ----          ----

          Input Date to be Displayed ->> 22-MAR

                                  Message
<RETURN>-Exit   <REMOVE>-Break
```

```
          ID  Device Program     Act Date          Time
          --  --------------     --- ----          ----
          43  28LADC0848100      UP  22-MAR-1990   08:30:48
          41  68LHPC46083130     DN  22-MAR-1990   11:31:58
          40  68LHPC46083130     DN  22-MAR-1990   11:41:53
          39  68LSCX6B120EGB10   UP  22-MAR-1990   11:55:26
          37  68LSCXHPC4608310   ER  22-MAR-1990   13:04:00
          37  68LHPC46083130     DN  22-MAR-1990   13:05:51
          38  68LC01713Q         DN  22-MAR-1990   13:42:39
          40  68LHPC46083130     UP  22-MAR-1990   13:43:51
          50  44LSCX6206ABY130   UP  22-MAR-1990   13:45:04
          43  28LSCX6B10AEQ100   ER  22-MAR-1990   13:56:38
          43  28LSCX6B10AEQ100   UP  22-MAR-1990   13:58:18
          37  68LC0171130        ER  22-MAR-1990   21:06:45
          38  68LC017130         UP  22-MAR-1990   21:08:31
                                  Message
Press <RETURN> for More .....
<RETURN>-Exit   <REMOVE>-Break
```

```
        .file "vsii_create.asm"

#############################################################################
# VSII CREATE
#                                   PRIORITY        TASK ID
#       TASK:    1. PXI_PORT        9               30
#                2. PXI_BXFER       10              32
#                3. XPC_BXFER       10              20
#                4. PXI_TIMER       12              38
#                5. XPC_TIMER       12              24
#                6. PXI_MSGTX       16              36
#                7. PXI_MSGRX       18              34
#                8. XPC_MSGRX       18              22
#                9. PXI_MON         30              40
#               10. User task       xx              xx
#               11. User task       xx              xx
#               12. User task       xx              xx
#
#                                                   QUEUE ELEMENTS
#       QUEUES:  1. QUE_PXI_PORT                    1500
#                2. QUE_ACE3                        8
#                3. QUE_SW1                         2
#                4. QUE_PXI_RSND                    SND_COUNT
#                5. QUE_PXI_MSGRX                   RCV_COUNT
#                6. QUE_PXI_MSGTX                   SND_COUNT
#                7. QUE_SEC_COMM                    1000
#                8. QUE_SEC_MSGRX                   SEC_COUNT
#                9. User queue                      xx
#               10. User queue                      xx
#
# Note: The maximum allocated queues and elements reserved in the VRTX
#       workspace is 10 queues and a total of 4744 queue elements for
#       both VSII and PROGRAM usage. If this is to be exceeded, the VRTX
#       workspace (VRTX_WS) has to be recalculated.
#############################################################################

#include "vsii_equ.inc"

.text

vsii_create::

        bsr     xpc_init

        bsr     pxi_create      # create pxi communication tasks
```

```
        movd    $SCFTCREATE,r0
        addr    xpc_bxfer,r1
        movd    $(10),r2                # task priority
        movd    $(20),r3                # task id
        movqd   0,r4
        svc

# create task : ( PRI = 18, ID = 22, user mode)

        movd    $SCFTCREATE,r0
        addr    xpc_msgrx,r1
        movd    $(18),r2                # task priority
        movd    $(22),r3                # task id
        movqd   0,r4
        svc

# create task : ( PRI = 12, ID = 24, user mode)

        movd    $SCFTCREATE,r0
        addr    xpc_timer,r1
        movd    $(12),r2                # task priority
        movd    $(24),r3                # task id
        movqd   0,r4
        svc

# create queue comm as receive buffer - ace_bd1 task (ACE 11 & 18)

        movd    $SCFQCREATE,r0
        movd    $(QUE_SEC_COMM),r2
        movd    $(1000),r3
        svc

# create queue msgrx for message received from port

        movd    $SCFQCREATE,r0
        movd    $(QUE_SEC_MSGRX),r2
        movd    $(SEC_COUNT),r3
        svc

# create queue ace 2 (not used)

#@      movd    $SCFQCREATE,r0
#@      movd    $(QUE_ACE2),r2
#@      movd    $(1000),r3
#@      svc
```

```
        .file   "xpc_bxfer.asm"

#include "vsii_equ.inc"

        .text

 #-------------------------------------------------------------------
 # IDLE: This section waits for data on QUE_SEC_COMM. The data received
 # is then decoded for a character and port id.
 # Content of r3 : [xx-xx-port_id-char]    xx=don't care
 #-------------------------------------------------------------------
xpc_bxfer::
idle:
        movqd   $(0),r3                 # indefinite wait for data[r3]
        movd    $(QUE_SEC_COMM),r2      # from any ace port.
        movd    $SCFQPEND,r0
        svc
        cmpd    r0,$(0)                 # QUE_SEC_COMM error ?
        bne     idle                    # yes

 #-------------------------------------------------------------------
 # The following are derived from the information in Register R3.
 # It represents the basic information from one of the ACE port.
 #-------------------------------------------------------------------
        movzbd  r3,r2           # R2 = Input data.
        lshw    $(-8),r3        #
        movzbd  r3,r1           # R1 = Port ID
        bsr     get_pcb_addr    # R4 = PCB Addr
                                # or
        cmpd    r4,$(-1)        # invalid port id
        beq     idle

        movqd   $(0),PCB_TIMER(r4)       # reset timeout timer.
        movd    $POLL_TIME,POL_TIMER(r4) # reset polling timer

        movd    COM_STATE(r4),r5         # R5 = COMM State..
        bsr     case_state
        br      idle

 #-------------------------------------------------------------------
 # This section executes the routine for the different comm_state.
 # r2 = Character       r4 = Port Control Block (PCB)
 #-------------------------------------------------------------------

case_state: cased       STBL[r5:d]
```

```
        .double state2-case_state       # Receive      - Length Byte
        .double state3-case_state       # Receive      - Message and Checksum
        .double state4-case_state       # Line Control - Receive EOT
        .double state5-case_state       # Send         - Send Byte
        .double state6-case_state       # Line Control - Send EOT
        .double state7-case_state       # Completion   - Error - send NAK
        .double state8-case_state       # Completion   - Receive - send ACK
        .double state9-case_state       # Completion   - Send - receive ACK/NAK
        .double state10-case_state      # Presence Polling Sequence.
        .double state11-case_state      # Equipment OFLINE Initialization.

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
# IDLE - MACHINE OFLINE (State = 0)
#        Begin communication only if Host is 'ONLNE'
#
# r2 = ENQ or STX       r4 = Port Control Block
#-------------------------------------------------------------------------
state0:
        cmpd    @HOST_STATUS,$(1)       # Is host system 'ONLINE'.
        beq     state1                  # YES.
        ret     0                       # NO - Disregard request.

#<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
# IDLE - MACHINE ONLINE (State = 1)
# r2 = ENQ or STX or POL        r4 = Port Control Block
#-------------------------------------------------------------------------
state1:
        cmpb    r2,$ENQ                 # Incoming message request.
        beq     state6                  # Line Control State.

        cmpb    r2,$POL                 # Request for Presence Polling ?
        beq     presence_poll_req       # YES

        cmpb    r2,$STX                 # Outgoing message request.
        bne     state1_ret

#-------------------------------------------------------------------------
# Outgoing message request.

        movqd   $(4),COM_STATE(r4)      # Set next state = 4
        movd    SEND_PTR(r4),r3         # get message addr

        movw    SEC_BLK(r3),r7          # Verify if message is first
        rotw    $(8),r7                 # block. If true, then save
        andw    $(0x7fff),r7            # address is Pointer.
```

```
send_enq:
        movd    @T2_BUFF,PCB_TIMER(r4)  # Set Protocol timeout value.
        movqb   $ENQ,r2                 #
        bsr     put_acex                # Send ENQ, r4 = PCB

state1_ret:
        ret     0

 #-------------------------------------------------------------------------
presence_poll_req:
        movd    $(10),COM_STATE(r4)     # Set next state = 10
        br      send_enq

 #-------------------------------------------------------------------------
 # RECEIVE - LENGTH BYTE
 # r2 = Length Byte     r4 = Port Control Block
 #-------------------------------------------------------------------------
state2:
        cmpb    r2,$ENQ                 # Equipment resend ENQ ?
        beq     state6_a                # YES - Send EOT again.

        cmpb    r2,$(10)                # Verify length byte received.
        blo     state7                  # Error - COMPLETION.

        addqd   $(2),r2                 # add 2 bytes for checksum byte
        movd    r2,RX_LENGTH(r4)        # save total length byte to receive

        movqd   $(3),COM_STATE(r4)      # Set next state  = 3
        movd    @T1_BUFF,PCB_TIMER(r4)  # Set intercharacter timeout value.
        ret     0

 #-------------------------------------------------------------------------
 # RECEIVE - MESSAGE BYTE and CHECKSUM
 # r2 = message/checksum        r4 = Port Control Block
 #-------------------------------------------------------------------------
state3:
        cmpd    RX_LENGTH(r4),$(2)      # Last 2 bytes are checksum bytes.
        ble     rx_checksum             # Not to be included in calculation.

        movd    RECD_PTR(r4),r3         # Get message address.
        addqb   $(1),LEN_BYTE(r3)       # Increment length byte.
        movzbd  LEN_BYTE(r3),r6         # Get next byte location.
        movb    r2,r3[r6:b]             # Store INPUT.
        addw    r2,RX_CHKSUM(r4)        # Calculate checksum.
```

```
        ret     0

rx_checksum:
        movd    RX_LENGTH(r4),r6                # Receives the two checksum
        subd    $(1),r6                         # bytes and store it in
        movb    r2,MSG_CHKSUM(r4)[r6:b]         # MSG_CHKSUM to compare with
                                                # the calculated CHECKSUM.
        acbd    $(-1),RX_LENGTH(r4),state3_ret  # Last byte ?

        cmpw    RX_CHKSUM(r4),MSG_CHKSUM(r4)    # YES - Compare checksum.
        bne     state7                          # Checksum mis-matched.
        br      state8                          # Checksum matched.

 #-------------------------------------------------------------------------
 # LINE CONTROL - RECEIVE EOT / ENQ (contention)
 # r2 = Character       r4 = Port Control Block
 #-------------------------------------------------------------------------
state4:
        cmpb    r2,$ENQ                 # CONTENTION ?
        beq     state6                  # YES - Send EOT.

        cmpb    r2,$EOT                 # Received EOT ?
        bne     retry                   # NO - Timeout (CAN.)

        movqw   $(0),TX_CHKSUM(r4)      # Clear checksum buffer.
        movqd   $(1),BYTE_POS(r4)       # Start byte position.

        movd    SEND_PTR(r4),r3         # Get message addr.
        movzbd  LEN_BYTE(r3),r2         # Get length byte.

        bsr     put_acex                # Send length byte, r4 = PCB

        addqd   $(2),r2                 # Add 2 count for checksum bytes.
        movd    r2,TX_LENGTH(r4)        # Save total bytes to send.
        movqd   $(5),COM_STATE(r4)      # Set state to 5 for auto sent.

  # enable the port for transmit empty interrupt

        movd    PCB_PORT(r4),r1         # The data are sent using
        addr    ace_addr_bd1,r7         # interrupt mode.
        movd    r7[r1:d],r6             #
        movqb   $(3),int_enable_reg(r6) # Enable tx_empty interrupt
        ret     0

 #-------------------------------------------------------------------------
```

```
#-----------------------------------------------------------------------
state5:
        ret     0

#-----------------------------------------------------------------------
# LINE CONTROL - SEND EOT
# r2 = Character        r4 = Port Control Block
#-----------------------------------------------------------------------
state6:
        bsr     get_sec_BUF             # Find free buffer space.
        cmpd    r3,$(-1)                # Check error code.
        beq     state6_ret              # No more buffer - don't send EOT.

        movd    r3,RECD_PTR(r4)         # Save buffer address.
        movqb   $(0),LEN_BYTE(r3)       # Clear length byte.

state6_a:
        movd    $EOT,r2                 #
        bsr     put_acex                # Send EOT, r4 = PCB.

        movqd   $(2),COM_STATE(r4)      # WAIT for length byte
        movd    @T2_BUFF,PCB_TIMER(r4)  # Set intercharacter timeout value.

state6_ret:
        ret     0

#-----------------------------------------------------------------------
# COMPLETION - ERROR
# r2 = Character        r4 = Port Control Block
#-----------------------------------------------------------------------
state7:
        movqd   $(7),COM_STATE(r4)      # completion - error - send NAK
        movd    @T1_BUFF,PCB_TIMER(r4)  # Set intercharacter timeout value.
        ret     0

#-----------------------------------------------------------------------
# COMPLETION - RECEIVE SUCCESS - SEND ACK
# r2 = Character        r4 = Port Control Block
#-----------------------------------------------------------------------
state8:
        movd    $ACK,r2                 #
        bsr     put_acex                # send ACK, r4 = PCB

        addqd   $(1),MSG_RX_COUNT(r4)   # *diag - rx message count
```

```
        movqd   $(0),RECD_PTR(r4)       # clear address pointer

        movd    PCB_PORT(r4),PORT_ID(r3) # tag port id to end of secs message
        movqd   $(0),NBLOCK_POINTER(r3)  #

        movd    $QUE_SEC_MSGRX,r2        # and post message to tsk_msgrx
        movd    $SCFQPOST,r0             # through QUE_SEC_MSGRX
        svc                              #

        cmpd    r0,$(0)                  # Post successful ?
        beq     message_posted           # YES

        bsr     sec_discard              # Unable to post to MSGRX task
        br      check_tx_pend            # Discard message just received.

message_posted:
        movqd   $(0),RX_LENGTH(r4)       # clear receive length byte buffer
        movqw   $(0),MSG_CHKSUM(r4)      # clear receive checksum buffer
        movqw   $(0),RX_CHKSUM(r4)       # clear cal. checksum buffer

        cmpb    PCB_MID(r4),$(0)         # unattached port
        beq     check_tx_pend

 # check if equipment has changed.

        movb    SEC_TYP(r3),r0
        andb    $(0x7f),r0
        cmpb    r0,PCB_TYP(r4)
        bne     equip_swap
        cmpb    SEC_MID(r3),PCB_MID(r4)
        beq     check_tx_pend

 # Detected attached equipment id is different. (Changes or Swapped)
equip_swap:
        movd    $(11),COM_STATE(r4)      # ofline init
        ret     0

 #-------------------------------------------------------------------
 # COMPLETION - SEND - RECEIVE ACK / NAK
 # r2 = Character       r4 = Port Control Block
 #-------------------------------------------------------------------
state9:
        cmpb    r2,$ACK                  # Message sent successful ?
        bne     retry                    # NO - Retry.
```

```
        movd    NBLOCK_POINTER(r3),SEND_PTR(r4) # load new block address

        tbitb   $(7),SEC_BLK(r3)        # last of block ?
        bfc     check_tx_pend           # no

# end of message
        movqd   $(0),NBLOCK_POINTER(r3) # clear last block pointer buff
        movd    FIRST_BLOCK(r4),r3      # load first block address
        movqd   $(0),FIRST_BLOCK(r4)    # clear first block buffer

        tbitb   $(7),SEC_STM(r3)        # W-bit set ?
        bfc     delete_message          # NO

# message requires a reply

        bsr     mark_expected_block     # r3 = msg addr
        cmpd    r0,$(0)                 # r1 = return status.
        beq     check_tx_pend

# message does not require a reply
delete_message:
        bsr     sec_discard

#-------------------------------------------------------------------------
check_tx_pend:
        movqd   $(3),POL_RETRY(r4)      # reset presence poll retry
        cmpb    PCB_TYP(r4),$(0)        # new port ?
        bne     check_tx_pend_a         # no

# new port
        movqd   $(0),COM_STATE(r4)      #yes
        cmpd    @HOST_STATUS,$(0)       # host ofline ?
        bne     check_tx_pend_b         # yes - stop send

# new port and host is 'dead'

        bsr     equip_ofline_a          # clear port control block
        br      check_ret

check_tx_pend_a:
        movqd   $(1),COM_STATE(r4)      # idle - machine online

check_tx_pend_b:
        cmpd    SEND_PTR(r4),$(0)       # send message pending
        beq     check_ret
```

```
        movd    PCB_PORT(r4),r3         # load port id
        lshw    $(8),r3
        movb    $STX,r3                 # load request to send (STX)
        movd    $QUE_SEC_COMM,r2
        movd    $SCFQPOST,r0
        svc

check_ret:
        ret     0

 #-------------------------------------------------------------------------
 # PRESENCE POLLING STATE 10 - RECEIVE EOT / ENQ (contention)
 # r2 = Character        r4 = Port Control Block
 #-------------------------------------------------------------------------
state10:
        cmpb    r2,$ENQ         # received ENQ
        beq     state6          # CONTENTION - Equip request to send

        cmpb    r2,$EOT         # received EOT
        beq     check_tx_pend

  # pstate_retry - r2 = CAN or other character

        acbd    $(-1),POL_RETRY(r4),resend_poll
        bsr     equip_ofline            #
        ret     0

 resend_poll:
        movd    PCB_PORT(r4),r3
        lshw    $(8),r3
        movb    $POL,r3
        movd    $QUE_SEC_COMM,r2
        movd    $SCFQPOST,r0
        svc

        movqd   $(1),COM_STATE(r4)
        ret     0

   #-------------------------------------------------------------------------
   # this a dummy state to allow the equipment ofline init to function
   # without any message coming in or going out. It is reseted to zero(0)
   # once ofline init is completed.

  state11::
        ret     0
```

```
#############################################################################
retry:
        movd    SEND_PTR(r4),r3
        addqd   $(1),RTY_COUNT(r4)      # *diag - number of retries done
        acbd    $(-1),SEC_RTY(r3),check_tx_pend

# retry expire
        cmpb    PCB_TYP(r4),$(0)        # New port ?
        beq     reset_port              # Zero state polling.

        cmpd    COM_STATE(r4),$(4)      # equipment ofline ?
        beq     port_ofline             # protocol_timeout

#---------------------------------------------------------------------------
# retrieve message (single or multiblock) for sec_recover.
retrieve:
        tbitb   $(7),SEC_BLK(r3)        # Last block of message ?
        bfs     last_block              # YES
        movd    NBLOCK_POINTER(r3),r3   # Get next message block.
        br      retrieve

last_block:
        movd    NBLOCK_POINTER(r3),SEND_PTR(r4) # Load new message addr.
        movd    $(0),NBLOCK_POINTER(r3)         # Mark end of block.
        movd    FIRST_BLOCK(r4),r3              # load 1st block address
        movqd   $(0),FIRST_BLOCK(r4)            # Clear first block pointer.
        bsr     sec_recover
        br      check_tx_pend

#---------------------------------------------------------------------------
# Equipment has gone ofline.
port_ofline:
        bsr     equip_ofline
        ret     0

#---------------------------------------------------------------------------
# No equipment attached during zero state poll.
reset_port:
        bsr     equip_ofline_a
        ret     0
```

```
        .file   "xpc_msgrx.asm"

#include "vsii_equ.inc"

        .text

xpc_msgrx::
msgrx_idle:
        movqd   $(0),@SEC_PRIMARY       # reset primary message marker.
        movqd   $(0),r3                 # indefinite wait on QUE_SEC_MSGRX
        movd    $QUE_SEC_MSGRX,r2       # for any incoming messages [r3]
        movd    $SCFQPEND,r0
        svc
        cmpd    r0,$(0)                 # queue error ?
        bne     msgrx_idle              # yes

 #-----------------------------------------------------------------------
        movd    PORT_ID(r3),r1  # R1 = Port id.
                                # R3 = Message address.

        bsr     get_pcb_addr    # return: r4 = pcb addr
        cmpd    r4,$(-1)        #       or r4 = -1 (invalid port id)
        beq     deallocate      #

 ########################################################################
 # <+KNOWN DEVICE+>                                                     #
 # check for the followings :                                           #
 #                                                                      #
 #      1. UNSUPPORTED TYPE (OPERATION)                                 #
 #      2. NEW MACHINE ATTACHED                                         #
 #      3. MACHINE ID CHANGED                                           #
 #                                                                      #
 #      r3 = message address, r4 = PCB                                  #
 ########################################################################

        movzbd  SEC_TYP(r3),r6                  # r6 = machine type
        andb    $(0x7f),r6                      # remove r-bit
        cmpb    r6,@type_switch
        bne     unsupported_type                # unsupported device type

        cmpb    PCB_TYP(r4),$(0)                # new machine attached ?
        beq     new_machine_attached            # yes

        cmpb    SEC_MID(r3),PCB_MID(r4)         # id mismatch ?
        bne     machine_change                  # yes
```

```
# check for duplicate message block ( identical message header)        #
# r3 = message address, r4 = PCB                                       #
#########################################################################

check_duplicate_block::

        cmpmb   1(r3),SEC_HEADER(r4),0x0a       # compare header
###     beq     deallocate                      # duplicate message
        movmb   1(r3),SEC_HEADER(r4),0x0a       # transfer new header

#########################################################################
# <+EXPECTED+>                                                          #
# Begin expected block check                                            #
# r3 = message address, r4 = PCB                                        #
#########################################################################

        addr    EXB_BUFFER(r4),r0       # expected block address
        movd    $EXB_COUNT,r7           # expected block limit

nxt_exb::
        cmpd    EXB_TIMER(r0),$(0)      # empty EXB block ?
        beq     nxt_exb_a               # yes
        cmpd    SEC_SYS(r3),EXB_SYSBYT(r0) # system byte = expected ?
        bne     nxt_exb_a               # no

        cmpd    SEC_SYS(r3),$(0)        # system bytes = 0000
        bne     secondary               # no

        tbitb   $(0),SEC_FNC(r3)        # primary message ?
        bfc     nxt_exb_a               # no

 # Only Equipment initiated multiblock primary message with all their syst
 # bytes = 0000 will go through this section. Need to conduct more check t
 # verify that the message is of the expected stream & function.

        movd    EXB_ADDR(r0),r6         # get previous expected message
        cmpb    SEC_STM(r3),SEC_STM(r6) # same stream ?
        bne     nxt_exb_a               # no
        cmpb    SEC_FNC(r3),SEC_FNC(r6) # same function ?
        beq     secondary               # yes

nxt_exb_a:
        addd    $(EXB_SIZE),r0
        acbd    $(-1),r7,nxt_exb
```

```
# r3 = message address, r4 = PCB                                        #
##########################################################################

        tbitb   $(0),SEC_FNC(r3)
        bfc     deallocate              # unexpected secondary message

##########################################################################
# <+FIRST BLOCK - PRIMARY MESSAGE+>                                      #
# Check if block is FIRST BLOCK of secondary message                     #
# r3 = message address, r4 = PCB                                         #
##########################################################################

        movw    SEC_BLK(r3),r7          # check block number
        andw    $(0xff7f),r7
        rotw    $(8),r7
        cmpw    r7,$(0)                 # block number = 0 ?
        beq     last_block
        cmpw    r7,$(1)                 # block number = 1 ?
        bne     deallocate              # not 1st block

##########################################################################
# <+LAST BLOCK+>
# Last block of Primary or Secondary message
# r3 = message address, r4 = PCB
##########################################################################

last_block::
        tbitb   $(7),SEC_BLK(r3)        # last block ??
        bfs     secsII_message_complete # yes.

##########################################################################
# [+SET INTER-BLOCK TIMER / SET EXPECTED BLOCK+]
# MULTIBLOCK (MTB) MESSAGE - ********
# Note:
# Support one(1) multiblock message per port at any one time.
# Any existing uncompleted multiblock message will be deleted once
# a new multiblock message is received
##########################################################################

        cmpd    MTB_PTR(r4),$(0)
        beq     begin_mtb
        save    [r3]
        movd    MTB_PTR(r4),r3
        bsr     sec_discard             # deallocate old MTB message
        restore [r3]
```

```
        movw    SEC_BLK(r3),SEC_BLK_NUM(r4)
        rotw    $(8),SEC_BLK_NUM(r4)
        addw    $(1),SEC_BLK_NUM(r4)    # increment count by one
        rotw    $(8),SEC_BLK_NUM(r4)

set_interblock_timer:
        bsr     mark_expected_block
        cmpd    r0,$(0)
        beq     msgrx_idle              # (+WAIT FOR NEXT BLOCK+)

 #-------------------------------------------------------------------------
 # Error - No more space to mark expected block.
 # Discard Multiblock message sequence.
 #-------------------------------------------------------------------------
        movd    MTB_PTR(r4),r3
        bsr     sec_discard
        br      msgrx_idle

 ###########################################################################
 # <+SECONDARY+>
 # Expected block
 # r0 = PCB_EXB, r3 = message address, r4 = PCB
 ###########################################################################

secondary:
        movqd   $(0),EXB_SYSBYT(r0)     # reset system byte
        movqd   $(0),EXB_TIMER(r0)      # reset timer
        movqd   $(0),EXB_ADDR(r0)       # reset prev. msg address pointer
        tbitb   $(0),SEC_FNC(r3)        # secondary message ?
        bfs     add_block               # no. add block to primary message

 ###########################################################################
 # <+FIRST BLOCK - SECONDARY MESSAGE+>
 # Check if block is FIRST BLOCK of secondary message
 # r3 = message address, r4 = PCB
 ###########################################################################

        movd    SEC_SYS(r3),@SEC_PRIMARY        # get primary message addr
        movw    SEC_BLK(r3),r7                  # check block number
        andw    $(0xff7f),r7
        rotw    $(8),r7
        cmpw    r7,$(0)                 # block number = 0
        beq     last_block              # or
        cmpw    r7,$(1)                 # block number = 1
        beq     last_block
```

```
# r3 = message address, r4 = PCB
#########################################################################

add_block::     # MTB

# check message block count. Must be one higher than the previous block.

        movw    SEC_BLK(r3),r7
        andw    $(0xff7f),r7
        cmpw    SEC_BLK_NUM(r4),r7
        bne     block_error
        rotw    $(8),SEC_BLK_NUM(r4)
        addw    $(1),SEC_BLK_NUM(r4)    # increment count by one
        rotw    $(8),SEC_BLK_NUM(r4)

        movd    MTB_PTR(r4),r0          # get multiblock start address
nxt_mtb:
        cmpd    NBLOCK_POINTER(r0),$(0) # end of multi_block
        beq     end_mtb                 # YES
        movd    NBLOCK_POINTER(r0),r0   # get next block address
        br      nxt_mtb

end_mtb:
        movd    r3,NBLOCK_POINTER(r0)   # save address in previous block

 ##########################################################################
 # <+LAST BLOCK - MULTIBLOCK+>                                            #
 # r3 = message address, r4 = PCB                                         #
 ##########################################################################

        tbitb   $(7),SEC_BLK(r3)        # message completed ??
        bfc     set_interblock_timer    # no

 # multiblock_message_completed
mtb_complete:

        movd    MTB_PTR(r4),r3          # get 1st block address
        movqd   $(0),MTB_PTR(r4)        # clear MTB secs block pointer

 ##########################################################################
 # [+PRIMARY or SECONDARY MESSAGE COMPLETE+]
 # Begin message validation procedure.
 # r3 = message address, r4 = PCB
 ##########################################################################
```

```
        movb    SEC_STM(r3),r6          # r6[b] = stream byte
        cbitb   $(7),r6                 # remove rbit

 # Check if message is supported.
verify_sf:
        cmpw    0(r7),$(-1)             # end of table search (-1)?
        beq     unsupported_message     #
        cmpb    0(r7),r6                # verify stream byte
        bne     next_sf                 # invalid stream
        movqd   $(1),@SEC_STREAM_FLAG   # mark stream flag
        cmpb    1(r7),SEC_FNC(r3)       # verify function byte
        beq     valid_sf                #

next_sf:
        addd    $(TAB_SIZE),r7          # get next S/F address.
        br      verify_sf

 #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
 # JUMP TO USER DEFINE SERVICE ROUTINE
 #       r3 = message address
 #       r4 = PCB address
 #       r5 = Equipment state

valid_sf:
        save    [r3,r4]                 # save message and PCB address
        movd    EQP_STATE(r4),r5        # r5 = Message state
        movd    2(r7)[r5:d],r6          # load supporting action routine
        jsr     r6                      # call user action routine
        restore [r3,r4]
 #
 #^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

 ##########################################################################
 # this section will deallocate the secs II message after processing.

deallocate::
        bsr     sec_discard
        cmpd    @SEC_PRIMARY,$(0)       # primary message address
        beq     msgrx_idle

  # deallocate primary message

        movd    @SEC_PRIMARY,r3
        bsr     sec_discard
        br      msgrx_idle
```

```
# check for duplicate type and id
# r3 = message address, r4 = PCB
#########################################################################

new_machine_attached::

        cmpb    SEC_MID(r3),$(0)
        beq     illegal_dev_id                  # invalid device id

        movb    PCB_PORT(r4),@PORTN             # save new port id
        addr    PCB_BUFFER,r5
        movqd   $(1),r7                         #

check_duplicate_id::
        cmpb    SEC_MID(r3),PCB_MID(r5)         # same machine id ?
        beq     duplicate_id                    # yes
        addd    $PCB_SIZE,r5                    #
        addd    $(1),r7                         # next port
        cmpb    r7,$PCB_COUNT                   # End of table ?
        ble     check_duplicate_id              # no

#-------------------------------------------------------------------------
# valid new machine attached

        movb    SEC_TYP(r3),PCB_TYP(r4)         # save machine TYPE and
        andb    $(0x7f),PCB_TYP(r4)             # ID on the PCB
        movb    SEC_MID(r3),PCB_MID(r4)         #
        save    [r3]
        movb    SEC_MID(r3),r0                  # get device id
        bsr     s151f167                        # Report equip online
        restore [r3]
        bsr     port_startup                    # send s1f1 to equip
        addqd   $(1),@EQ_COUNT                  # increment equipment coun
        br      check_duplicate_block

#########################################################################
# Machine attached to port has changed. Report to Host that the former
# equipment has gone ofline. Clear the Port Control Block. Discard the
# current message from the newly attached machine. Zero state polling
# will startup the newly attached machine.
# r3 = message address, r4 = PCB
#########################################################################

machine_change::
        bsr     equip_ofline                # clear port control block
```

```
unsupported_type:
        cmpb    PCB_TYP(r4),$(0)        # new machine attached ?
        bne     machine_change          # yes

 #xxx   bsr     s9f1                    # received different type
        bsr     s159f101                # Report illegal type
        br      deallocate

 #-------------------------------------------------------------------
 # New machine just attached to port has id = Zero(0)
 # r3 = message address, r4 = PCB
 #-------------------------------------------------------------------

illegal_dev_id:
 #xxx   bsr     s9f1                    # received id = zero(0)
        bsr     s159f101                # report illegal id
        br      deallocate

 #-------------------------------------------------------------------
 # New machine just attached to port has duplicate id
 # r3 = message address, r4 = PCB
 #-------------------------------------------------------------------

duplicate_id:
 #xxx   bsr     s9f1
        bsr     s159f165                # report duplicate id
        br      deallocate

 #-------------------------------------------------------------------
unsupported_message:
        cmpb    @SEC_STREAM_FLAG,$(0)           # stream flag set ?
        beq     unsupported_function    # no
        bsr     s9f3                    # unrecognized stream type
        bsr     s159f103                # report unsupported stream
        br      deallocate

 unsupported_function:
        bsr     s9f5                    # unrecognized function type
        bsr     s159f105                # Report unsupported function
        br      deallocate

  #-------------------------------------------------------------------
  # Receive Multiblock messages block number incorrectly.
```

```
bsr     sec_discard
movd    MTB_PTR(r4),r3
movqd   $(0),MTB_PTR(r4)
br      deallocate
```

```
        .file   "xpc_timer.asm"

#include "vsii_equ.inc"

        .text

xpc_timer::
timer_idle:
        movd    $TIMER_COUNT,r2         # 1 clock tick - 1ms
        movd    $SCFTDELAY,r0
        svc
 ##########################################################################
 # Begin checking individual port timer counters.
 #-------------------------------------------------------------------------

        addr    PCB_BUFFER,r4           # r4 = PCB address
        movqd   $(1),r1                 # r1 = port id

 #-------------------------------------------------------------------------
 # SECS-I Communication Timer
 #-------------------------------------------------------------------------
comm_timer:
        cmpd    PCB_TIMER(r4),$(0)      # timer set ?
        beq     poll_timer              # no

        acbd    $(-1),PCB_TIMER(r4),poll_timer  # decrement TIMER buffer

 # timeout occur ----

        movd    COM_STATE(r4),r5        # Load Timeout State.
        bsr     timeout_state           # Timeout action routine.

 #-------------------------------------------------------------------------
 # Poll timer looks at both Zero State and Presence polling.
 #-------------------------------------------------------------------------
poll_timer:
        acbd    $(-1),POL_TIMER(r4),exb_timer
        bsr     polling_sequence

 #-------------------------------------------------------------------------
 # EXB  timer looks at timeout waiting for reply or multiblock message.
 #-------------------------------------------------------------------------
exb_timer:
        bsr     exb_message_check

 #-------------------------------------------------------------------------
```

```
next_PCB:
        addd    $PCB_SIZE,r4            # next PCB block
        addqd   $(1),r1                 # next port id
        cmpd    r1,$PCB_COUNT           # last port ?
        ble     comm_timer              # no.
        br      timer_idle              # finish cycle

 ##########################################################################
 # POLL / POLL / POLL / POLL / POLL / POLL / POLL / POLL / POLL / POLL
 ##########################################################################
polling_sequence::
        save    [r0,r1,r2,r3,r4,r7]
        movd    $POLL_TIME,POL_TIMER(r4)
        cmpd    COM_STATE(r4),$(0)      # unattached port ?
        beq     zero_state_poll

 #-------------------------------------------------------------------------
presence_poll::
        cmpd    SEND_PTR(r4),$(0)
        bne     polling_ret
        cmpd    COM_STATE(r4),$(1)
        bne     polling_ret
        movb    $POL,r2
        br      poll_send

 #-------------------------------------------------------------------------
zero_state_poll::
        cmpd    @HOST_STATUS,$(0)       # host ofline
        beq     polling_ret

 # Build a S1F1 message with Equipment ID = 0 ****

        movb    $(1),r1                 # stream 1
        movb    $(1),r2                 # function 2
        movqd   $(0),r7                 # wbit = 1
        bsr     sec_bldhdr
        cmpd    r3,$(-1)
        beq     polling_ret

        cmpd    SEND_PTR(r4),$(0)       # message pending ?
        beq     zstate_poll             # no
        save    [r3]                    # save current message
        movd    SEND_PTR(r4),r3         # load old message address
        movqd   $(0),SEND_PTR(r4)       # Reset pointer.
        bsr     sec_discard             # discard message
```

```
        cmpd    COM_STATE(r4),$(0)      # bxfer active ?
        beq     zstate_poll_b           # no

zstate_poll_a:
        bsr     sec_discard             # discard message
        br      polling_ret

zstate_poll_b:
        movd    r3,SEND_PTR(r4)
        movb    $STX,r2

poll_send:
        movd    PCB_PORT(r4),r3
        lshw    $(8),r3
        movb    r2,r3
        movd    $(QUE_SEC_COMM),r2
        movd    $SCFQPOST,r0
        svc
        cmpd    r0,$(0)
        bne     zstate_poll_a

polling_ret:
        restore [r0,r1,r2,r3,r4,r7]
        ret     0

 ##########################################################################
 # EXPECTED / EXPECTED / EXPECTED / EXPECTED / EXPECTED / EXPECTED
 ##########################################################################
exb_message_check::
        save    [r0,r3,r7]
        addr    EXB_BUFFER(r4),r0
        movd    $EXB_COUNT,r7

exb_a:
        cmpd    EXB_TIMER(r0),$(0)              # expecting message ?
        beq     exb_d                           # no
        acbd    $(-1),EXB_TIMER(r0),exb_d       # yes - decrement by 1 sec

 # expected block timeout occur

        addqd   $(1),T3_COUNT(r4)
        movd    EXB_ADDR(r0),r3                 # load message addr

        movqd   $(0),EXB_TIMER(r0)              # clear timer counter
        movqd   $(0),EXB_SYSBYT(r0)             # clear system byte
```

```
 # from equipment to host. [rbit set = E->H message]

        tbitb   $(7),SEC_TYP(r3)                # check r-bit: set?
        bfs     exb_b                           # yes

        bsr     sec_recover
        br      exb_d

 # incomplete equipment to host multiblock message

exb_b:
        cmpd    MTB_PTR(r4),$(0)
        beq     exb_c
        movd    MTB_PTR(r4),r3
exb_c:
        bsr     sec_discard

 # get next expected block address
exb_d:
        addd    $EXB_SIZE,r0
        acbd    $(-1),r7,exb_a
        restore [r0,r3,r7]
        ret     0

 #>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
timeout_state::
        cased   TTBL[r5:d]

TTBL::  .double s0_timeout-timeout_state
        .double s1_timeout-timeout_state
        .double s2_timeout-timeout_state
        .double s3_timeout-timeout_state
        .double s4_timeout-timeout_state
        .double s5_timeout-timeout_state
        .double s6_timeout-timeout_state
        .double s7_timeout-timeout_state
        .double s8_timeout-timeout_state
        .double s9_timeout-timeout_state
        .double s10_timeout-timeout_state

 #<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

s0_timeout::    # no action taken. Timeout should not happen in this state
s1_timeout::    # no action taken. Timeout should not happen in this state
```

```
s2_timeout::    #  timeout waiting for length byte. Abort receive procedur

###     addqd   $(1),T1_COUNT(r4)       # increment T1 timeout count
        br      recd_abort

#************************************************************************

s3_timeout::    # intercharacter timeout. abort and goto state 7

        addqd   $(1),T1_COUNT(r4)       # increment T1 timeout count
        movqd   $(7),COM_STATE(r4)      # error completion
        movd    @T1_BUFF,PCB_TIMER(r4)
        ret     0

#************************************************************************

s4_timeout::    # protocol timeout waiting for EOT. Abort and retry.

        addqd   $(1),T2_COUNT(r4)
        br      send_abort

#************************************************************************

s5_timeout::    # no action taken. Timeout should not happen in this state

        ret     0

#************************************************************************

s6_timeout::    # no action taken. Timeout should not happen in this state

        ret     0

#************************************************************************

s7_timeout::    # Intercharacter timeout - completion error
        addqd   $(1),T1_COUNT(r4)
        movd    $NAK,r2
        bsr     put_acex
        br      recd_abort

#************************************************************************

s8_timeout::    # no action taken. Timeout should not happen in this state
```

```
#*************************************************************************

s9_timeout::    # protocol timeout waiting for ACK or NAK. Abort and retry

        addqd   $(1),T2_COUNT(r4)
        br      send_abort

#*************************************************************************

s10_timeout::   # presence poll timeout waiting for EOT. Abort and retry.

        addqd   $(1),T2_COUNT(r4)

#*************************************************************************

send_abort:
        movzbd  r1,r3                   # get port id
        lshw    $(8),r3                 # upper word = port id
        movb    $CAN,r3                 # lower word = CAN
        movd    $QUE_SEC_COMM,r2
        movd    $SCFQPOST,r0
        svc
        ret     0

#*************************************************************************

recd_abort:
        movd    RECD_PTR(r4),r3
        movqd   $(0),RECD_PTR(r4)       # clear pointer
        bsr     sec_discard

        movqd   $(0),RX_CHKSUM(r4)      # clear rx checksum
        movqd   $(0),MSG_CHKSUM(r4)     # clear message checksum
        movqd   $(0),RX_LENGTH(r4)      # clear rx_length
        movqd   $(1),COM_STATE(r4)      # return to idle state

        cmpb    PCB_TYP(r4),$(0)        # new port ?
        bne     check_send              # no
        bsr     equip_ofline_a          # reset port

 # check for message pending for transmission

check_send:
        cmpd    SEND_PTR(r4),$(0)       # message pending ?
        beq     check_ret               # no
```

```
        lshw    $(8),r3
        movb    $(STX),r3               # request xpc_block_xfer to
        movd    $QUE_SEC_COMM,r2        # begin send procedure.
        movd    $SCFQPOST,r0
        svc

check_ret:
        ret     0
```

```
        .file   "xpc_rout.asm"

#include "vsii_equ.inc"

        .text
 ########################################################################
 # Initialize memory area
 ########################################################################

xpc_init::

 # For diagnostic purpose. To check on buffer address

        movd    $SEC_BUFFER,r0
        movd    $SEC_MEM_SIZE,r0

        bsr     clear_all_pcb
        bsr     clear_sec_block

 # Set all timeout timers and retry (T1,T2,T3,T4 & RTY) to defaul

        movd    $SEC_T1,@T1_BUFF
        movd    $SEC_T2,@T2_BUFF
        movd    $SEC_T3,@T3_BUFF
        movd    $SEC_T4,@T4_BUFF
        movd    $SEC_RETRY,@RTY_BUFF
        movqd   $(0),@EQ_COUNT
        movqd   $(0),@USE_BUF_COUNT

        ret     0

 ########################################################################
 # clear all secs message block ********************

clear_sec_block::
        save    [r4,r5,r7]
        addr    SEC_BUFFER,r4
        movd    $SEC_COUNT,r5

next_secs_block:
        movd    $SEC_SIZE,r7
        bsr     clear_block

        addd    $SEC_SIZE,r4
```

```
        ret     0

 ######################################################################
 # clear all PCB - Program Control Block **********

clear_all_pcb::
        save    [r1,r4]
        addr    PCB_BUFFER,r4
        movd    $(1),r1
next_PCB:
        movd    r1,PCB_PORT(r4)         # set pcb_id
        bsr     clear_pcb
        addd    $PCB_SIZE,r4
        addd    $(1),r1
        cmpd    r1,$PCB_COUNT
        ble     next_PCB
        restore [r1,r4]
        ret     0

 ######################################################################
 # Subroutine: clear_PCB - resets the Port Control Block to Zero.
 # r4 = PCB address

clear_pcb::
        save    [r1,r4,r7]
        movd    $(PCB_SIZE),r7
        movd    PCB_PORT(r4),r1         # save pcb_id
        bsr     clear_block
        movd    r1,PCB_PORT(r4)         # restore pcb_id

 # this section sets the MODEL and SOFTRV buffer to <dash>

        movqd   $(0),r1
dash:
        movb    $(0x2d),MODEL(r4)[r1:b]
        addqb   $(1),r1
        cmpb    r1,$(12)
        blt     dash

 # this section sets the PCB initial default value.

        movd    $SEC_RETRY,@POL_RETRY
        movd    $(15),POL_TIMER(r4)     # default time (15 sec)fo
                                        # zero state poll
        restore [r1,r4,r7]
```

```
# Subroutine to clear memory
# r4 = memory address  r7 = number of bytes to clear

clear_block::
        subd    $(1),r7
clear:
        movb    $(0),0(r4)[r7:b]
        acbd    $(-1),r7,clear
        movb    $(0),0(r4)[r7:b]
        ret     0

#=================================================================
# Input : r3 = secs message, r4 = PCB
# return: r0 = 0 (success), -1 (fail)

mark_expected_block::

        save    [r1]
        addr    EXB_BUFFER(r4),r0
        movd    $EXB_COUNT,r1

check_block::

        cmpd    EXB_ADDR(r0),$(0)
        beq     store_syswrd
        addd    $(EXB_SIZE),r0          # next block address
        acbd    $(-1),r1,check_block
        movqd   $(-1),r0
        restore [r1]
        ret     0

store_syswrd::
        movd    @T3_BUFF,EXB_TIMER(r0)          # set expected ti
        movd    SEC_SYS(r3),EXB_SYSBYT(r0)      # save system byt
        movd    r3,EXB_ADDR(r0)                 # save msg addr
        movqd   $(0),r0                         # success
        restore [r1]
        ret     0

#=================================================================
# input: r1 = port id  return: r4 = pcb address

get_pcb_addr::
        cmpb    r1,$(0)
        beq     invalid_port_id
```

```
        save    [r1]
        addr    PCB_BUFFER,r4
search:
        acbb    $(-1),r1,next_block
        restore [r1]
        ret     0

next_block:
        addd    $PCB_SIZE,r4
        br      search

invalid_port_id:
        movd    $(-1),r4
        ret     0

 ######################################################################
 #                                                                    #
 #       SUBROUTINE:     verify_eqid                                  #
 #                                                                    #
 #       input : r3 = pxj message address                             #
 #       output: r4 = PCB address                                     #
 #               r7 = - 1 - eqid  not found                           #
 #                                                                    #
 ######################################################################

verify_eqid::
        addr    PCB_BUFFER,r4
        movd    $(1),r7

verify_eqid_a:
        cmpd    COM_STATE(r4),$(0)
        beq     next_eqid
        cmpb    PXI_MID(r3),PCB_MID(r4)
        beq     eqid_found

next_eqid:
        addd    $PCB_SIZE,r4
        addqd   $(1),r7
        cmpd    r7,$PCB_COUNT
        ble     verify_eqid_a

invalid_eqid:
        movd    $(-1),r7
eqid_found:
        ret     0
```

```
#       SUBROUTINE:     sec_recover                                     #
#                                                                       #
#       input: r3 = sec message address                                 #
#########################################################################

sec_recover::
        save    [r3,r4,r5,r6,r7]
        addr    PCB_BUFFER,r4
        movd    $PCB_COUNT,r7

sec_recover_a:
        cmpd    COM_STATE(r4),$(0)
        beq     sec_recover_b
        cmpd    COM_STATE(r4),$(11)
        beq     sec_recover_b
        cmpb    SEC_MID(r3),PCB_MID(r4)
        beq     sec_recover_c

sec_recover_b:
        addd    $PCB_SIZE,r4
        acbd    $(-1),r7,sec_recover_a
        br      disregard_message
sec_recover_c:
        movb    SEC_STM(r3),r6          # Save Stream byte
        cbitb   $(7),SEC_STM(r3)        # clear R-bit

        addr    error_table,r7          # r4 = PCB address

sec_recover_d:
        cmpw    0(r7),$(-1)             # end of table
        beq     disregard_message

        cmpb    0(r7),SEC_STM(r3)
        bne     sec_recover_e
        cmpb    1(r7),SEC_FNC(r3)
        beq     sec_recover_f

sec_recover_e:
        addd    $(TAB_SIZE),r7
        br      sec_recover_d

 # S/F found. Extract Action routine address.

sec_recover_f:
        movb    r6,SEC_STM(r3)          # restore stream byte
```

```
        save    [r0,r1,r2,r3,r4,r5]
        jsr     r6                      # goto user action routin
                                        # r3 = message addr.
                                        # r4 = PCB addr.
                                        # r7 = ** (Return Status)
        restore [r0,r1,r2,r3,r4,r5]
        cmpd    r7,$(0)                 # Discard message ?
        beq     sec_recover_ret         # NO.

disregard_message:
        bsr     sec_discard

sec_recover_ret:
        restore [r3,r4,r5,r6,r7]
        ret     0

 ###################################################################
 #                                                                 #
 #      SUBROUTINE:     Duplicate EQID (Vax level)                 #
 #                                                                 #
 #              r3 = message address                               #
 #                                                                 #
 ###################################################################
duplicate_eqid::
        save    [r3,r4,r7]

        addr    PCB_BUFFER,r4
        movd    $PCB_COUNT,r7

duplicate_eqid_a:
        cmpb    18(r3),PCB_MID(r4)
        beq     duplicate_eqid_b

        addd    $PCB_SIZE,r4
        acbd    $(-1),r7,duplicate_eqid_a
        br      duplicate_ret

duplicate_eqid_b:
        subd    $(1),@EQ_COUNT
        bsr     equip_ofline_a

duplicate_ret:
        restore [r3,r4,r7]
        ret     0
```

```
#                                                                    #
#              r3 = message address                                  #
#              r4 = PCB address                                      #
#              FIRST_BLOCK(r4) = first block address                 #
#                                                                    #
######################################################################

equip_ofline::
        movb    PCB_MID(r4),r0
        bsr     s151f169                # report_eqoff
        subd    $(1),@EQ_COUNT

 # begin clearing all messages related to this equipment.
equip_ofline_a::
        save    [r0,r3,r4,r7]
        movd    $(11),COM_STATE(r4)     # ofline init state

 # this section clears all messages pointed by FIRST_BLOCK
        cmpd    FIRST_BLOCK(r4),$(0)
        beq     equip_ofline_b
        movd    FIRST_BLOCK(r4),r3
        movqd   $(0),FIRST_BLOCK(r4)
        bsr     sec_discard

 # this section clears all messages pointed by SEND_PTR
equip_ofline_b::
        cmpd    SEND_PTR(r4),$(0)       # message pending ?
        beq     equip_ofline_c          # NO
        movd    SEND_PTR(r4),r3         # Discard message.
        movqd   $(0),SEND_PTR(r4)       # If message already disc
        bsr     sec_discard             # r3 will be returned = -

 # this section clears all expected message
equip_ofline_c:
        addr    EXB_BUFFER(r4),r0
        movd    $EXB_COUNT,r7

equip_ofline_d:
        cmpd    EXB_TIMER(r0),$(0)
        beq     equip_ofline_e

        movqd   $(0),EXB_TIMER(r0)      # reset timer
        movd    EXB_ADDR(r0),r3         # load expected message a
        movqd   $(0),EXB_ADDR(r0)
        movqd   $(0),EXB_SYSBYT(r0)
```

```
        addd    $EXB_SIZE,r0            # next expected block
        acbd    $(-1),r7,equip_ofline_d

 # this section clears all messages pointed by MTB_PTR
        cmpd    MTB_PTR(r4),$(0)
        beq     equip_ofline_f
        movd    MTB_PTR(r4),r3
        movqd   $(0),MTB_PTR(r4)
        bsr     sec_discard

 # this section clear the message pointed by the RECD_PTR
equip_ofline_f:
        cmpd    RECD_PTR(r4),$(0)
        beq     equip_ofline_g
        movd    RECD_PTR(r4),r3
        movqd   $(0),RECD_PTR(r4)
        bsr     sec_discard

 # this section clear all status, pointer in the equipment PCB
equip_ofline_g:
        bsr     clear_pcb               # r4 = PCB address

        restore [r0,r3,r4,r7]
        ret     0


 ######################################################################
 #                                                                    #
 #       SUBROUTINE:     pxi_diag_req (s152,f25)                      #
 #                                                                    #
 #               r4 = PCB address                                     #
 #       output: r7 = nil                                             #
 #                                                                    #
 ######################################################################

xpc_diag_req::
        cmpb    PXI_MID(r3),$(0)        # message for XPC
        bne     xpc_diag_secs           # no
        bsr     s152f25
        br      xpc_diag_ret

xpc_diag_secs:
        save    [r0,r1,r2,r3,r7]
        movb    $(2),r1         # Stream 2
        movb    $(25),r2        # Func. 25
```

```
        bsr     sec_sndmsg
        restore [r0,r1,r2,r3,r7]
 #00    bsr     pxi_sec_send
xpc_diag_ret:
        ret     0

rx_s2f26::
        save    [r0,r1,r2,r3,r7]
        movb    $(152),r1       # Stream 152
        movb    $(26),r2        # Func. 26
        movqd   $(0),r7         # wbit = 0
        movd    r3,r0
        bsr     sec_pxi_rebuild
        bsr     pxi_sndmsg
        restore [r0,r1,r2,r3,r7]
        ret     0

 ###################################################################
 #                                                                 #
 #       SUBROUTINE:     pxi_diag_data (s152,f26)                  #
 #                                                                 #
 #       input : r3 = pxi message address                          #
 #               r4 = PCB address                                  #
 #       output: r7 = nil                                          #
 #                                                                 #
 ###################################################################

xpc_diag_dat::
        cmpb    PXI_MID(r3),$(0)        # message for XPC
        beq     xpc_data_ret            # yes
        bsr     pxi_sec_send
xpc_data_ret:
        ret     0

 ###################################################################
 #                                                                 #
 #       SUBROUTINE:     xpc_date_time                             #
 #                                                                 #
 #       input : r3 = pxi message address                          #
 #               r4 = PCB address                                  #
 #       output: r7 = nil                                          #
 #                                                                 #
 ###################################################################

xpc_date_time::
```

```
        br      xpc_date_time_ret

mc_date_time:
        bsr     pxi_sec_send
xpc_date_time_ret:
        ret     0

 ###################################################################
 #                                                                 #
 #       subroutine : S1F1                                         #
 #       this subroutine builds a stream 1 function 2 message      #
 #       r3 = message address                                      #
 #       r4 = PCB                                                  #
 ###################################################################

s1f1::

 # begin building reply message header - r2 = primary message add

        save    [r0,r1,r3]
        movd    r3,r0
        jsr     sec_rephdr
        cmpd    r3,$(-1)
        beq     s1f1_return

 # host exception - build a zero length list structure

        movqb   $(0),r1
        jsr     sec_bldlst
        cmpd    r3,$(-1)
        beq     s1f1_return

 # pass message block to SENDER

        jsr     sec_sndmsg

s1f1_return:
        restore [r0,r1,r3]
        ret     0

 ###################################################################

port_startup::
        save    [r1,r2,r3,r7]
        movb    $(1),r1         # Stream 1
```

```
        cmpd    r3,$(-1)
        beq     port_startup_ret
        bsr     sec_sndmsg
port_startup_ret:
        restore [r1,r2,r3,r7]
        ret     0


equip_online::
        movmb   15(r3),MODEL(r4),0x06
        movmb   23(r3),SOFTRV(r4),0x06
        ret     0

 ######################################################################
 #                                                                    #
 #      subroutine : S2F17                                            #
 #      this subroutine builds a stream 1 function 2 message          #
 #      r3 = message address                                          #
 #      r4 = PCB                                                      #
 ######################################################################

s2f17::
        bsr     sec_pxi_send
        ret     0

 ######################################################################
 #      System Error                                                  #
 #                                                                    #
 #      SUBROUTINE:     s9f1 (Illegal Equipment Type/ID)              #
 #      SUBROUTINE:     s9f3 (Unsupported Equipment Stream)           #
 #      SUBROUTINE:     s9f5 (Unsupported Equipment Functn)           #
 #                                                                    #
 #      this subroutine builds a stream 9 function 1,3,5 message      #
 #      r3 = message address                                          #
 #      r4 = PCB address                                              #
 ######################################################################

s9f1::
        movw    $(1),r2
        br      s9fx
s9f3::
        movw    $(3),r2
        br      s9fx
s9f5::
        movw    $(5),r2
```

```
s9fx:
        save    [r1,r3,r7]
        movb    $(9),r1
        movd    $(0),r7         # wbit = 0
        movd    r3,r0

        bsr     sec_bldhdr
        cmpd    r3,$(-1)
        beq     s9fx_return

 # build item #1

        addd    $(1),r0
        movb    $(0x21),r1
        movd    $(10),r7
        movd    r0,r2
        bsr     sec_blditm
        cmpd    r3,$(-1)
        beq     s9fx_return

 # pass message block to SENDER

        bsr     sec_sndmsg

s9fx_return:
        restore [r1,r3,r7]
        ret     0

 ######################################################################
 #                                                                    #
 #       SUBROUTINE:     s151f165 (Equipment List)                    #
 #                                                                    #
 #       input:          nil                                          #
 #       output:         nil                                          #
 #                                                                    #
 ######################################################################

 s151f165::
        save    [r0,r1,r2,r4,r6,r7]

        addr    PCB_BUFFER,r4
        movd    $PCB_COUNT,r6
        movqd   $(0),@EQ_COUNT
 equip_attach:
        cmpd    COM_STATE(r4),$(0)              # ofline
```

```
        addqd   $(1),@EQ_COUNT
        br      nxt_equip
ofl_equip:
        movqd   $(0),COM_STATE(r4)
        movqb   $(0),PCB_MID(r4)
nxt_equip:
        addd    $PCB_SIZE,r4
        acbd    $(-1),r6,equip_attach

        cmpd    @EQ_COUNT,$(0)          # equipment attached ?
        beq     skip                    # no

        movb    $(151),r1               # stream 151
        movb    $(165),r2               # function 165
        movqd   $(0),r7                 # wbit = 0
        bsr     pxi_bldhdr
        cmpd    r3,$(-1)
        bne     s151f165_a

 # Error - SND_Buffer full.             #
        bsr     backup_buffer           # Use Backup buffer
        bsr     pxi_bldhdr_x            # to send the message

s151f165_a:
        movd    @EQ_COUNT,r1            # build list structure
        bsr     pxi_bldlst
        cmpd    r3,$(-1)
        beq     skip

        addr    PCB_BUFFER,r4
        movd    $PCB_COUNT,r6
lp_port:
        cmpb    PCB_MID(r4),$(0)
        beq     nxt_port
        movb    $(0x21),r1              # data format + length-bi
        movqd   $(1),r7                 # data length
        addr    PCB_MID(r4),r2          # data address
        bsr     pxi_blditm
        cmpd    r3,$(-1)
        beq     skip
nxt_port:
        addd    $PCB_SIZE,r4
        acbd    $(-1),r6,lp_port
        bsr     pxi_sndmsg
skip:
```

```
#################################################################
#                                                               #
#       SUBROUTINE:     s151f167 (Equipment Online)             #
#       SUBROUTINE:     s151f169 (Equipment Ofline)             #
#                                                               #
#       input:          r0 = SEC_MID (Online)                   #
#       input:          r0 = PCB_MID (Ofline)                   #
#       output:         nil                                     #
#                                                               #
#################################################################

s151f167::
        save    [r0,r1,r2,r3,r7]
        movb    $(167),r2               # function 167
        br      s151f16x

s151f169::
        save    [r0,r1,r2,r3,r7]
        movb    $(169),r2               # function 169

s151f16x::
        movb    $(151),r1               # stream 151
        movqd   $(0),r7                 # wbit = 0
        bsr     pxi_bldhdr
        cmpd    r3,$(-1)
        bne     s151f16x_a

 # Error - SND_Buffer full.             #
        bsr     backup_buffer           # Use Backup buffer
        bsr     pxi_bldhdr_x            # to send the message

s151f16x_a:
        movb    r0,PXI_MID(r3)
        bsr     pxi_sndmsg
s151f16x_ret:
        restore [r0,r1,r2,r3,r7]
        ret     0

 #################################################################
 #      System Error                                             #
 #                                                               #
 #      SUBROUTINE:     s159f101 (Illegal Equipment Type/ID)     #
 #      SUBROUTINE:     s159f103 (Unsupported Equipment Stream)  #
 #      SUBROUTINE:     s159f105 (Unsupported Equipment Functn)  #
 #      SUBROUTINE:     s159f165 (Duplicate Equipment ID)        #
```

```
#       output:         nil                                             #
#                                                                       #
#########################################################################

s159f101::
        save    [r0,r1,r2,r3,r7]
        movb    $(101),r2
        br      s159f1xx

s159f103::
        save    [r0,r1,r2,r3,r7]
        movb    $(103),r2               # function 103
        br      s159f1xx

s159f105::
        save    [r0,r1,r2,r3,r7]
        movb    $(105),r2               # function 105
        br      s159f1xx

s159f165::
        save    [r0,r1,r2,r3,r7]
        movb    $(165),r2               # function 165

s159f1xx::
        movd    r3,r0   # save original message in r0
        movb    $(159),r1               # stream 159
        movqd   $(0),r7                 # wbit = 0
        bsr     pxi_bldhdr
        cmpd    r3,$(-1)
        beq     s159f1xx_ret            #

 # build list structure of 2
        movd    $(2),r1
        bsr     pxi_bldlst
        cmpd    r3,$(-1)
        beq     s159f1xx_ret            #

 # build item # 1 - MCID
        movb    $(0x21),r1      # data format + length-bit
        movqd   $(1),r7         # data length
        addr    SEC_MID(r0),r2  # data address
        bsr     pxi_blditm
        cmpd    r3,$(-1)
        beq     s159f1xx_ret
```

```
        addr    PCB_PORT(r4),r2 # data address
        bsr     pxi_bldltm
        cmpd    r3,$(-1)
        beq     s159f1xx_ret
        bsr     pxi_sndmsg

s159f1xx_ret:
        restore [r0,r1,r2,r3,r7]
        ret     0
```

```
        .file "xpc_bldmsg.asm"

 ##########################################################################
 # SECS MESSAGE BUILDER SUBROUTINES AND OTHER RELATED SUBROUTINES      #
 # FOR XPC_GEIC 32000 BOARD.                                           #
 ##########################################################################

#include "vsii_equ.inc"

        .text

 ########################################################################
 #                                                                      #
 #       SUBROUTINE: sec_bldhdr (build header)                          #
 #                                                                      #
 #       INPUT:                                                         #
 #               r1[7:0] = stream                                       #
 #               r2[7:0] = function                                     #
 #               r7[0:0] = wbit  00 or 01                               #
 #               r4      = PCB address                                  #
 #       OUTPUT:                                                        #
 #               r3 = msg_addr (First SECS block address)               #
 #          or   r3 =  -1  (error or no free block available)           #
 #                                                                      #
 ########################################################################

sec_bldhdr::
        bsr     get_sec_BUF                     # search for empty block
        cmpd    $(-1),r3                        # empty block found (r3)
        bne     sec_bldhdr_x                    #
        ret     0                               # no buffer available

 # set length byte to 10 byte and read TYPE and ID switch
sec_bldhdr_x::
        movb    $(0x0a),LEN_BYTE(r3)            # set length byte = 10
        movb    @type_switch,SEC_TYP(r3)        # get type
        movb    PCB_MID(r4),SEC_MID(r3)         # get id

 # set the stream and function byte
        movb    r1,SEC_STM(r3)                  # stream
        movb    r2,SEC_FNC(r3)                  # function

 # set the wbit bit according to parameter in r2(b)
        andb    $(0x01),r7                      # check w-bit
        rotb    $(7),r7                         # shift bit 1 to bit 8
```

```
# set system byte to message address.
        movd    r3,SEC_SYS(r3)                  #
        ret     0

#####################################################################
#                                                                   #
#       SUBROUTINE : sec_rephdr (build reply header)                #
#                                                                   #
#       INPUT:                                                      #
#               r0 = Primary message                                #
#               r4 = PCB address                                    #
#       OUTPUT:                                                     #
#               r3 = msg_addr (First SECS block address)            #
#          or   r3 = -1   (error or no free block available)        #
#####################################################################

sec_rephdr::
        movd    r0,r3
        bsr     verify_sec_addr                 # verify primary addr
        cmpd    $(-1),r3
        beq     sec_rephdr_ret                  # invalid primary addr
        bsr     get_sec_BUF                     # search for empty block
        cmpd    $(-1),r3                        # empty block found (r3)
        bne     sec_rephdr_x                    # yes
        ret     0                               # no buffer.

sec_rephdr_x::
        movb    $(0x0a),LEN_BYTE(r3)            # set length byte = 10
        movmb   1(r0),1(r3),0x0a                # copy header to builder b
        cbitb   $(7),SEC_TYP(r3)                # clear r-bit (H->E)
        cbitb   $(7),SEC_STM(r3)                # clear wait(w) bit
        addqb   $(1),SEC_FNC(r3)                # increment function by 1
        movw    $(0x0180),SEC_BLK(r3)           # set block count = 8001
sec_rephdr_ret:
        ret     0

#####################################################################
#                                                                   #
#       SUBROUTINE : sec_bldlst (build list structure)              #
#                                                                   #
#       INPUT:                                                      #
#               r1 = number of list (LST_NUM)                       #
#               r3 = msg_addr1 (First SECS block address)           #
#               r4 = PCB address                                    #
#                                                                   #
#       OUTPUT:                                                     #
```

```
#                                                                    #
######################################################################

sec_bldlst::
        bsr     verify_sec_addr
        cmpd    $(-1),r3
        bne     start_bldlst
        ret     0                               # invalid address

start_bldlst:
        save    [r3]                            # save msg_addr1
        bsr     current_block                   # find current secs block
        cmpd    $(-1),r3                        # block available ??
        beq     bldlst_error                    # No

 # build list format

        save    [r1]
        movb    $(1),r1
        bsr     save_byte
        restore [r1]
        cmpd    $(-1),r3                        # block available ??
        beq     bldlst_error                    # NO

 # build list count

        bsr     save_byte
        cmpd    $(-1),r3                        # block available ??
        beq     bldlst_error                    # NO
        restore [r3]
        ret     0

bldlst_error:
        restore [r3]                            # error return - insuffici
        bsr     sec_discard                     # memory
        ret     0                               #

 ######################################################################
 #                                                                    #
 #      SUBROUTINE : sec_blditm (build item)                          #
 #                                                                    #
 #      INPUT:                                                        #
 #              r1[7:0]  = format                                     #
 #              r2       = address where data is located              #
 #              r7[15:0] = size of data in byte                       #
```

```
#       OUTPUT:                                                 #
#               r3 = msg_addr1 (First SECS block address)       #
#          or   r3 =  -1  (error or no free block available)    #
#                                                               #
#################################################################

sec_blditm::
        bsr     verify_sec_addr
        cmpd    $(-1),r3
        bne     start_blditm
        ret     0                               # invalid address

start_blditm:
        save    [r3,r6,r7]
        bsr     current_block                   # find current secs block
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

        bsr     save_byte                       # save item format (r1)
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

        movzbd  r1,r6                           # save format byte
        andb    $(0x0fc),r6                     # clear bit 0/1 (length bi

        tbitb   $(1),r1                         # Format byte bit 1 set ?
        bfs     two_byte_len                    # YES.
        andw    $(0x00ff),r7
        br      one_byte_len

two_byte_len:
        movw    r7,r1
        rotw    $(8),r1
        bsr     save_byte                       # save item length
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

one_byte_len:
        movb    r7,r1
        bsr     save_byte                       # save item length
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

        cmpw    r7,$(0)                         # Data length = 0
        beq     end_blditm                      # YES - Done
```

```
        cmpb    r6,$(0x68)              # format = 2 byte integer
        beq     blditm_2byte            # signed
        cmpb    r6,$(0xa8)              # format = 2 byte integer
        beq     blditm_2byte            # unsigned

        cmpb    r6,$(0x70)              # format = 4 byte integer
        beq     blditm_4byte            # signed
        cmpb    r6,$(0x0b0)             # format = 4 byte integer
        beq     blditm_4byte            # unsigned
        cmpb    r6,$(0x090)             # format = 4 byte integer
        beq     blditm_4byte            # floating

#-------------------------------------------------------------------------
# For format 10(Binary), 11(Boolean, 20(Ascii), 30(8 byte signed integer)
#            40(8 byte floating point), 50(8 byte unsigned integer)

blditm_data:
        movb    0(r2),r1                # get data byte
        bsr     save_byte               # save data
        cmpd    $(-1),r3                # Error return ??
        beq     blditm_error            # yes
        addqd   $(1),r2                 # next data position
        acbb    $(-1),r7,blditm_data    # decrement length
        br      end_blditm              # data build completed

blditm_2byte:
        movb    1(r2),r1                # get data byte
        bsr     save_byte               # save 1st data
        cmpd    $(-1),r3                # Error return ??
        beq     blditm_error            # yes

        movb    0(r2),r1                # get data byte
        bsr     save_byte               # save 2nd data
        cmpd    $(-1),r3                # Error return ??
        beq     blditm_error            # yes

        addqd   $(2),r2                 # next data position
        acbb    $(-2),r7,blditm_2byte   # decrement length
        br      end_blditm              # data build completed

blditm_4byte:
        movb    3(r2),r1                # get data byte
        bsr     save_byte               # save data MSB
        cmpd    $(-1),r3                # Error return ??
        beq     blditm_error            # yes
```

```
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

        movb    1(r2),r1                        # get data byte
        bsr     save_byte                       # save data
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

        movb    0(r2),r1                        # get data byte
        bsr     save_byte                       # save data LSB
        cmpd    $(-1),r3                        # Error return ??
        beq     blditm_error                    # yes

        addqd   $(4),r2                         # next data position
        acbb    $(-4),r7,blditm_4byte           # decrement length

end_blditm:
        restore [r3,r6,r7]                      # data build completed
        ret     0

blditm_error:
        restore [r3,r6,r7]                      # error return - insuffici
        bsr     sec_discard
        ret     0                               # the message

 ###################################################################
 #                                                                 #
 #       SUBROUTINE : sec_sndmsg                                   #
 #                                                                 #
 #       INPUT:                                                    #
 #               r3 = 1st secs block address                       #
 #               r4 = PCB address                                  #
 #       OUTPUT:                                                   #
 #               r3(d) = current block address                     #
 #           or  r3(d) =  -1  (error or no free block available)   #
 #                                                                 #
 ###################################################################

sec_sndmsg::
        bsr     verify_sec_addr
        cmpd    $(-1),r3
        bne     sec_sndmsg_x
        ret     0                       # invalid address

sec_sndmsg_x:
```

```
        bne     message_pending         # yes

        movd    r3,SEND_PTR(r4)         # send message

        cmpd    COM_STATE(r4),$(1)      # port active ?
        bgt     sndmsg_ret

# port idle - request to send_message.

        movd    PCB_PORT(r4),r3         # get pcb port id
        lshw    $(8),r3                 # upper word = port id
        movb    $STX,r3                 # lower word = STX
        movd    $(QUE_SEC_COMM),r2
        movd    $SCFQPOST,r0
        svc                             # begin message sent

        br      sndmsg_ret

# this section store the next message to be sent if port
# is currently busy.

message_pending::

        movd    SEND_PTR(r4),r7
last_msg:
        cmpd    $(0),NBLOCK_POINTER(r7) # last message
        beq     append
        movd    NBLOCK_POINTER(r7),r7
        br      last_msg

append: movd    r3,NBLOCK_POINTER(r7)

sndmsg_ret:
        restore [r1,r2,r3,r4,r7]
        ret     0

 ###################################################################
 #                                                                 #
 #      SUBROUTINE : current_block                                 #
 #                                                                 #
 #      INPUT:                                                     #
 #              r3(d) = 1st secs block address                     #
 #      OUTPUT:                                                    #
 #              r3(d) = current block address                      #
 #          or  r3(d) =  -1  (error or no free block available)    #
```

```
current_block::
        cmpd    $(0),NBLOCK_POINTER(r3)
        beq     found_current
        movd    NBLOCK_POINTER(r3),r3   # get next block address.
        br      current_block

found_current:
        cmpb    $(254),LEN_BYTE(r3)
        blt     current_block_return
        bsr     new_block

current_block_return:
        ret     0

 ###################################################################
 #      SUBROUTINE: new_block                                      #
 #                                                                 #
 #      INPUT:                                                     #
 #            r3(d) = 1st secs block address                       #
 #            r4(d) = PCB                                          #
 #                                                                 #
 #      OUTPUT:                                                    #
 #            r3(d) = current block address                        #
 #         or r3(d) =  -1  (error or no free block available)      #
 #            r7(b) = current length byte                          #
 #                                                                 #
 ###################################################################

new_block::
        save    [r6]
        movd    r3,r6                   # transfer current address to r6
        bsr     get_sec_BUF             # search for empty block
        cmpd    $(-1),r3                # no free block ??.
        beq     new_block_return        # error return.

        movb    $(0x0a),LEN_BYTE(r3)    # set header byte count
        movd    r3,NBLOCK_POINTER(r6)   # store current address in prev bl
        movmb   1(r6),1(r3),0x0a        # transfer SECS header to new bloc
        andb    $(0x7f),SEC_BLK(r6)     # remove e-bit from previous block
        rotw    $(8),SEC_BLK(r3)        #
        addw    $(1),SEC_BLK(r3)
        rotw    $(8),SEC_BLK(r3)
new_block_return:
        restore [r6]
        ret     0
```

```
#       SUBROUTINE : get_sec_BUF                                        #
#                                                                       #
#       INPUT:                                                          #
#               r4 = PCB address                                        #
#       OUTPUT:                                                         #
#               r3(d) = block address                                   #
#           or  r3(d) = no free block available                         #
#                                                                       #
#########################################################################

get_sec_BUF::
        save    [r7]
        movd    $SEC_COUNT,r7           # max. number of secs block
nxt_BUF:
        movd    @SEC_BUF_CUR,r3                 # get current block
        addd    $SEC_SIZE,@SEC_BUF_CUR          # increment next block
        cmpd    @SEC_BUF_CUR,$SEC_MEM_SIZE      # end of block ?
        blt     chk_BUF                         # no
        addr    SEC_BUFFER,@SEC_BUF_CUR         # load start of block

chk_BUF:
        cmpd    $(0),0(r3)                      # blank block ?
        beq     found_BUF                       # yes
        acbd    $(-1),r7,nxt_BUF

        movd    $(-1),r3                        # no block available
        restore [r7]                            # full_BUF
        ret     0

found_BUF:
        movd    @RTY_BUFF,SEC_RTY(r3)           # set retry count
        movqd   $(0),SYSBYT_PXI(r3)             # pxi system byte
        movqw   $(0),APPLID_PXI(r3)             # pxi application id
        movqd   $(0),NBLOCK_POINTER(r3)         # mark last block
        movd    PCB_PORT(r4),PORT_ID(r3)        # mark port id

        addqd   $(1),@USE_BUF_COUNT             # *diag

        restore [r7]
        ret     0


#########################################################################
#                                                                       #
#       SUBROUTINE : save_byte                                          #
```

```
#               r3      = current block address                 #
#       OUTPUT:                                                 #
#               r3      = current or new block address          #
#           or  r3      =  -1  (no free block available)        #
#                                                               #
#################################################################

save_byte::
        cmpb    LEN_BYTE(r3),$(254)             # length byte = 254 ??
        blo     save_byte_a                     # no
        bsr     new_block                       # get new block
        cmpd    $(-1),r3                        # error return ??
        beq     save_byte_ret                   # yes
save_byte_a:
        save    [r7]
        addqb   $(1),LEN_BYTE(r3)               # increment length byte
        movzbd  LEN_BYTE(r3),r7
        movb    r1,r3[r7:b]                     # store byte
        restore [r7]
save_byte_ret:
        ret     0


 ################################################################
 #                                                              #
 #       SUBROUTINE : sec_discard                               #
 #                                                              #
 #       INPUT:                                                 #
 #               r3 = 1st secs block address                    #
 #       OUTPUT:                                                #
 #               r3 =  -1                                       #
 #                                                              #
 ################################################################
sec_discard::
        bsr     verify_sec_addr
        cmpd    $(-1),r3
        beq     sec_discard_ret

        movqd   $(0),0(r3)
        subd    $(1),@USE_BUF_COUNT
        movd    NBLOCK_POINTER(r3),r3
        cmpd    r3,$(0)
        bne     sec_discard

sec_discard_ret:
```

```
#####################################################################
#                                                                   #
#       SUBROUTINE : verify_sec_addr                                #
#                                                                   #
#       INPUT:                                                      #
#               r3 = secs block address                             #
#       OUTPUT:                                                     #
#               r3 = secs block address                             #
#           or  r3 = -1 (invalid secs block address)                #
#                                                                   #
#####################################################################

verify_sec_addr::
        cmpd    r3,$(-1)
        beq     invalid_addr
        cmpd    0(r3),$(0)
        beq     invalid_addr

        cmpd    r3,$SEC_BUFFER
        blt     invalid_addr
        cmpd    r3,$SEC_MEM_SIZE
        bgt     invalid_addr
        ret     0
invalid_addr:
        movqd   $(-1),r3
        ret     0
```

```
        .file   "xpc_cnvrt.asm"

#include "vsii_equ.inc"

        .text

 ######################################################################
 #                                                                    #
 #      SUBROUTINE : sec_pxi_rebuild (SECS -> PXI)                    #
 #                                                                    #
 #      DESCRIPTION: rebuild from secs to pxi message with the        #
 #                   specified Stream, Function and w-bit             #
 #                                                                    #
 #      INPUT:                                                        #
 #              r0(d) = SECS message address                          #
 #              r1(b)     = Stream                                    #
 #              r2(b)     = Function                                  #
 #              r7(bit 0) = w-bit                                     #
 #      OUTPUT:                                                       #
 #              r0(d) = SECS message address                          #
 #              r3(d) = PXI message address                           #
 #                   or -1   (error or no free block available)       #
 ######################################################################

sec_pxi_rebuild::
        save    [r0,r1,r2,r7]
        movd    r0,r3
        bsr     verify_sec_addr
        cmpd    $(-1),r3                        # valid sec message address
        beq     sec_pxi_rebuild_ret             # no

        bsr     get_SND_BUF                     # search for empty block
        cmpd    $(-1),r3                        # empty block found (r3)
        beq     sec_pxi_rebuild_ret             # no - r3 = -1

 ######################################################################
 #      Convert header (SECS -> PXI)                                  #
 ######################################################################

        movw    $(0x0e00),LEN_WORD(r3)          # set length byte = 14
        movw    @PIPID,PXI_PID(r3)              # set pip id
        movb    @id_switch,PXI_GID(r3)          # set geic id
        movqb   $(1),PXI_FLG(r3)                # set direction I->H
        movb    @type_switch,PXI_TYP(r3)        # set operation type
        movb    SEC_MID(r0),PXI_MID(r3)         # copy machine id
```

```
        tbitb   $(0),r7                         # reply required ?
        bfc     sec_pxi_rebuild_a               # NO
        sbitb   $(1),PXI_FLG(r3)                # YES - Set pxi reply bit

sec_pxi_rebuild_a:
        tbitb   $(0),PXI_FNC(r3)                # primary or secondary message?
        bfc     secondary_pxi_rebuild           #

 ##########################################################################
 #      Primary message (SECS -> PXI)                                     #
 ##########################################################################
        movd    SEC_SYS(r0),SYSBYT_SEC(r3)      # yes-save secs system byte
        movd    r3,PXI_SYS(r3)
        br      sec_pxi_rebuild_b

 ##########################################################################
 #      Secondary message (SECS -> PXI)                                   #
 ##########################################################################
secondary_pxi_rebuild:
        movd    SEC_SYS(r0),r7                  # get primary message address
        movw    APPLID_PXI(r7),PXI_AID(r3)      # copy pxi application id
        movd    SYSBYT_PXI(r7),PXI_SYS(r3)      # copy pxi system bytes

 ##########################################################################
 #      Copy SECS message body                                            #
 ##########################################################################
sec_pxi_rebuild_b:
        bsr     sec_pxi_copdat                  # r0 = secs message addr
                                                # r3 = pxi message addr
sec_pxi_rebuild_ret:                            #      or -1 error
        restore [r0,r1,r2,r7]
        ret     0

 ##########################################################################
 #                                                                        #
 #      SUBROUTINE : pxi_sec_rebuild  (PXI -> SECS)                       #
 #                                                                        #
 #      DESCRIPTION: rebuild from pxi to secs message with the            #
 #                   specified Stream, Function and w-bit                 #
 #                                                                        #
 #      INPUT:                                                            #
 #              r0(d) = PXI message address                               #
 #              r1(b) = Stream                                            #
 #              r2(b) = Function                                          #
 #              r7(bit 0) = w-bit                                         #
```

```
#              r3(d) = SECS message address                    #
#                   or -1   (error or no free block available) #
#################################################################

pxi_sec_rebuild::
        save    [r0,r1,r2,r7]

        movd    r0,r3                           #
        bsr     verify_pxi_addr                 #
        cmpd    $(-1),r3                        # Valid pxi message ?
        beq     pxi_sec_rebuild_ret             # NO

        bsr     get_sec_BUF                     # search for empty block
        cmpd    $(-1),r3                        # empty block found (r3)
        beq     pxi_sec_rebuild_ret             # no - r3 = -1

#########################################################################
#       Convert Header (PXI -> SECS)                                    #
#########################################################################
        movb    $(0x0a),LEN_BYTE(r3)            # set length byte = 10
        movb    @type_switch,SEC_TYP(r3)        # copy type
        movb    PXI_MID(r0),SEC_MID(r3)         # copy ID

        movb    r1,SEC_STM(r3)                  # r1 = message stream
        movb    r2,SEC_FNC(r3)                  # r2 = message function
        movw    $(0x0180),SEC_BLK(r3)           # set block count = 8001

        tbitb   $(0),r7                         # reply required ?
        bfc     pxi_sec_rebuild_a               # NO
        sbitb   $(7),SEC_STM(r3)                # YES - Set w-bit

pxi_sec_rebuild_a:
        tbitb   $(0),SEC_FNC(r3)                # primary or secondary message
        bfc     secondary_sec_rebuild

#########################################################################
#       Primary message (PXI -> SECS)                                   #
#########################################################################
        movd    PXI_SYS(r0),SYSBYT_PXI(r3)      # save  pxi system byte
        movw    PXI_AID(r0),APPLID_PXI(r3)      # save pxi application ID
        movd    r3,SEC_SYS(r3)
        br      pxi_sec_rebuild_b

#########################################################################
#       Secondary message (PXI -> SECS)                                 #
#########################################################################
```

```
        movd    SYSBYT_SEC(r7),SEC_SYS(r3)      # restore system byte

 ##########################################################################
 #      Copy PXI message body
 ##########################################################################
pxi_sec_rebuild_b:
        bsr     pxi_sec_copdat                  # r0 = pxi message addr
                                                # r3 = sec message addr
pxi_sec_rebuild_ret:                            #      or -1 error
        restore [r0,r1,r2,r7]
        ret     0

 #@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

 ########################################################################
 #                                                                      #
 #      SUBROUTINE : sec_pxi_send  (SECS -> PXI)                        #
 #                                                                      #
 #      DESCRIPTION: convert from secs to pxi message and               #
 #                   sends the message to Host.                         #
 #                                                                      #
 #      INPUT:                                                          #
 #           r3(d) = SECS message address                               #
 #      OUTPUT:                                                         #
 #           r3(d) = SECS message address                               #
 #         or  r7(d) = -1   (error or no free block available)          #
 ########################################################################

sec_pxi_send::
        save    [r0,r3]

        bsr     verify_sec_addr                 #
        cmpd    $(-1),r3                        # Valid sec message address ?
        beq     sec_pxi_send_error              # NO

        movd    r3,r0                           # R0 = SECS MESSAGE ADDRESS

        bsr     get_SND_BUF                     # search for empty block
        cmpd    $(-1),r3                        # empty block found (r3)
        beq     sec_pxi_send_error              # no - r3 = -1

 ##########################################################################
 #      Convert header. (SECS -> PXI)                                     #
 ##########################################################################

        movw    $(0x0e00),LEN_WORD(r3)          # set length byte = 14
```

```
        movqb   $(1),PXI_FLG(r3)            # set direction I->H
        movb    @type_switch,PXI_TYP(r3)    # set operation type
        movb    SEC_MID(r0),PXI_MID(r3)     # copy machine id

        movb    SEC_STM(r0),PXI_STM(r3)     # copy message stream
        cbitb   $(7),PXI_STM(r3)            # remove wbit
        addb    $(150),PXI_STM(r3)          # stream + 150
        tbitb   $(7),SEC_STM(r0)            # secs wbit set ?
        bfc     sec_pxi_send_a              # no
        sbitb   $(1),PXI_FLG(r3)            # set pxi reply bit

sec_pxi_send_a:
        movb    SEC_FNC(r0),PXI_FNC(r3)     # copy message function
        tbitb   $(0),PXI_FNC(r3)            # primary or secondary message?
        bfc     secondary_pxi               #

 ###########################################################################
 #      Primary message (SECS -> PXI)                                      #
 ###########################################################################
        movd    SEC_SYS(r0),SYSBYT_SEC(r3)  # yes-save secs system byte
        movd    r3,PXI_SYS(r3)
        br      sec_pxi_send_b

 ###########################################################################
 #      Secondary message (SECS -> PXI)                                    #
 ###########################################################################
secondary_pxi:
        movd    SEC_SYS(r0),r7              # get primary message address
        movw    APPLID_PXI(r7),PXI_AID(r3)  # copy pxi application id
        movd    SYSBYT_PXI(r7),PXI_SYS(r3)  # copy pxi system bytes

 ###########################################################################
 #      Copy SECS message body                                             #
 ###########################################################################
sec_pxi_send_b:
        bsr     sec_pxi_copdat              # r0 = secs message addr
        cmpd    $(-1),r3                    # r3 = pxi message addr
        beq     sec_pxi_send_error          #      or -1 error

 ###########################################################################
 #      Send PXI message                                                   #
 ###########################################################################

        bsr     pxi_sndmsg
        movqd   $(0),r7
```

```
sec_pxi_send_error:
        movqd   $(-1),r7
        restore [r0,r3]
        ret     0

 ######################################################################
 #                                                                    #
 #       SUBROUTINE : pxi_sec_send  (PXI -> SECS)                     #
 #                                                                    #
 #       DESCRIPTION: convert from pxi to secs message and            #
 #                    sends the message to the machine.               #
 #                                                                    #
 #       INPUT:                                                       #
 #               r3(d) = PXI message address                          #
 #               r4(d) = PCB                                          #
 #       OUTPUT:                                                      #
 #               r3(d) = PXI message address                          #
 #               r4(d) = PCB                                          #
 #          or   r7(d) = -1   (error or no free block available)      #
 ######################################################################

pxi_sec_send::
        save    [r0,r3]

        bsr     verify_pxi_addr             #
        cmpd    $(-1),r3                    # Valid pxi message ?
        beq     pxi_sec_send_error          # NO

        movd    r3,r0                       # R0 = PXI MESSAGE ADDRESS

        bsr     get_sec_BUF                 # search for empty block
        cmpd    $(-1),r3                    # empty block found (r3)
        beq     pxi_sec_send_error          # no - r3 = -1

 ########################################################################
 #       Convert Header (PXI -> SECS)                                   #
 ########################################################################
        movb    $(0x0a),LEN_BYTE(r3)        # set length byte = 10
        movb    PXI_TYP(r0),SEC_TYP(r3)     # copy type
        movb    PXI_MID(r0),SEC_MID(r3)     # copy ID

        movb    PXI_STM(r0),SEC_STM(r3)     # copy stream
        subb    $(150),SEC_STM(r3)          # stream - 150
        tbitb   $(1),PXI_FLG(r0)            # reply required ?
        bfc     pxi_sec_send_a              # no reply
```

```
        movw    $(0x0180),SEC_BLK(r3)           # set block count = 8001

        movb    PXI_FNC(r0),SEC_FNC(r3)         # copy function
        tbitb   $(0),SEC_FNC(r3)                # primary or secondary message
        bfc     secondary_sec

#######################################################################
#       Primary message (PXI -> SECS)                                 #
#######################################################################
        movd    PXI_SYS(r0),SYSBYT_PXI(r3)      # save  pxi system byte
        movw    PXI_AID(r0),APPLID_PXI(r3)      # save pxi applcation ID
        movd    r3,SEC_SYS(r3)
        br      pxi_sec_send_b

#######################################################################
#       Secondary message (PXI -> SECS)                               #
#######################################################################
secondary_sec:
        movd    PXI_SYS(r0),r7                  # load primary message address
        movd    SYSBYT_SEC(r7),SEC_SYS(r3)      # restore system byte

#######################################################################
#       Copy PXI message body
#######################################################################
pxi_sec_send_b:
        bsr     pxi_sec_copdat                  # r0 = pxi message addr
        cmpd    $(-1),r3                        # r3 = sec message addr
        beq     pxi_sec_send_error              #       or -1 error

#######################################################################
#       Send SECS message
#######################################################################
pxi_sec_send_c:
        bsr     sec_sndmsg
        movqd   $(0),r7
        restore [r0,r3]
        ret     0

pxi_sec_send_error:
        movqd   $(-1),r7
        restore [r0,r3]
        ret     0

#@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
        beq     sec_pxi_copdat_error    #
        movd    $(10),r7                # load sec byte position
        cmpb    r7,LEN_BYTE(r0)         #
        beq     sec_pxi_copdat_error    #
        br      sec_pxi_copdat_b        # continue data transfer

 # completed data transfer
sec_pxi_copdat_d:
        restore [r0,r3,r6,r7]
        rotw    $(8),r5
        movw    r5,LEN_WORD(r3)
        restore [r5]
        ret     0

sec_pxi_copdat_error:
        restore [r0,r3,r6,r7]
        restore [r5]
        bsr     pxi_discard
        movd    $(-1),r3
        ret     0

 ######################################################################
 #                                                                    #
 #       SUBROUTINE : pxi_sec_copdat                                  #
 #                                                                    #
 #       INPUT:                                                       #
 #               r0(d) = PXI message address                          #
 #               r3(d) = SEC msg_addr1 (first block address)          #
 #               r4(d) = PCB                                          #
 #                                                                    #
 #       OUTPUT:                                                      #
 #               r0(d) = PXI message address                          #
 #               r3(d) = SEC msg_addr1 (first block address)          #
 #          or   r3(d) =  -1  (error or no free block available)      #
 #                                                                    #
 ######################################################################

pxi_sec_copdat::
        save    [r1,r3,r5,r6]           # save msg_addr1
        movw    LEN_WORD(r0),r5         # get message length
        rotw    $(8),r5                 #
        subw    $(14),r5                # r5 = total data length
        cmpw    r5,$(0)                 # Header only message ?
        beq     pxi_sec_copdat_d        # YES - End of PXI message.
        movd    $(14),r6                # pxi header length
```

```
        movb    1(r0)[r6:b],r1          # get byte from PXI message
        bsr     save_byte               # save byte in SEC message
        cmpd    $(-1),r3                # block available ??
        beq     pxi_sec_copdat_error    # NO
        acbd    $(-1),r5,pxi_sec_copdat_c  # decrement total data length
        br      pxi_sec_copdat_d        # end of PXI message data

pxi_sec_copdat_c:
        cmpd    r6,$PXI_MAX             # end of PXI block ?
        blt     pxi_sec_copdat_b                # no
        movd    NBLOCK_CHAIN(r0),r0     # get next pxi block
        movd    $(0),r6                 # reset byte pointer
        br      pxi_sec_copdat_b

pxi_sec_copdat_d:
        restore [r1,r3,r5,r6]
        ret     0

pxi_sec_copdat_error:
        restore [r1,r3,r5,r6]           # error return - insufficient
        bsr     sec_discard             # memory
        ret     0                       #
```

```
        .file "xpc_acebd1.asm"

#include "vsii_equ.inc"

        .macro  save_all_registers
        movd    r0,tos                  # save r0 - for ui_exit use.
        save    [r1,r2,r3,r4,r5,r6,r7]
        .endm

 ########################################################################
 # This file contain the ace interrupt routine and put_ace routine      #
 # for all the eight ports of ACE Board 1.                              #
 ########################################################################

        .text

 #-----------------------------------------------------------------------#
 #  Character receive interrupt handler                                  #
 #  for port 11 ( INPUT TERMINAL)                                        #
 #-----------------------------------------------------------------------#

get_ace11::
        save_all_registers
        movd    $(0x0100),r3            # ace port id
        movd    $(ace_11),r6            # ace port address
        br      receive_data

 #-----------------------------------------------------------------------#
 #  Character receive interrupt handler                                  #
 #  for port 12 ( INPUT TERMINAL)                                        #
 #-----------------------------------------------------------------------#

get_ace12::
        save_all_registers
        movd    $(0x0200),r3            # ace port id
        movd    $(ace_12),r6            # ace port address
        br      receive_data

 #-----------------------------------------------------------------------#
 #  Character receive interrupt handler                                  #
 #  for port 13 ( INPUT TERMINAL)                                        #
 #-----------------------------------------------------------------------#

get_ace13::
        save_all_registers
```

```
#-------------------------------------------------------------------------#
#  Character receive interrupt handler                                    #
#  for port 14 ( INPUT TERMINAL)                                          #
#-------------------------------------------------------------------------#

get_ace14::
        save_all_registers
        movd    $(0x0400),r3            # ace port id
        movd    $(ace_14),r6            # ace port address
        br      receive_data

#-------------------------------------------------------------------------#
#  Character receive interrupt handler                                    #
#  for port 15 ( INPUT TERMINAL)                                          #
#-------------------------------------------------------------------------#

get_ace15::
        save_all_registers
        movd    $(0x0500),r3            # ace port id
        movd    $(ace_15),r6            # ace port address
        br      receive_data

#-------------------------------------------------------------------------#
#  Character receive interrupt handler                                    #
#  for port 16 ( INPUT TERMINAL)                                          #
#-------------------------------------------------------------------------#

get_ace16::
        save_all_registers
        movd    $(0x0600),r3            # ace port id
        movd    $(ace_16),r6            # ace port address
        br      receive_data

#-------------------------------------------------------------------------#
#  Character receive interrupt handler                                    #
#  for port 17 ( INPUT TERMINAL)                                          #
#-------------------------------------------------------------------------#

get_ace17::
        save_all_registers
        movd    $(0x0700),r3            # ace port id
        movd    $(ace_17),r6            # ace port address
        br      receive_data
```

```
#-------------------------------------------------------------------------#

get_ace18::
        save_all_registers
        movd    $(0x0800),r3            # ace port id
        movd    $(ace_18),r6            # ace port address

#########################################################################
# ACE interrupt routine (ACE xx)= r6 / data => r2 => QUE_SEC_COMM
# r3 = port id + data

receive_data:
        cmpb    $(4),int_ident_reg(r6)  # Receive Data Available ?
        bne     transmit_empty          # no - transmit empty interrupt

 # Receive data available

        movb    data_reg(r6),r3         # insert input character in lower byte

        movd    $(QUE_SEC_COMM),r2      # Post port_id & input to
        movd    $SCFQPOST,r0            # block xfer task
        svc

 # interrupt exit via VRTX

intrpt_exit:

        restore [r1,r2,r3,r4,r5,r6,r7]
        bispsrw $(0x0800)               # enable interrupts

        movd    $UIFEXIT,r0             # TR_EXIT fuction code
        svc

error:  bpt

 # Transmitter Holding Register Empty
transmit_empty::

        movd    r3,r1
        lshw    $(-8),r1                # input : r1 = port id
        bsr     get_pcb_addr            # return: r4 = pcb addr

        cmpd    $(5),COM_STATE(r4)      # is state = 5 ?
        beq     send_mess               # yes
        movqb   $(1),int_enable_reg(r6) # disable tx_empty interrupt
```

```
        cmpd    TX_LENGTH(r4),$(2)      # last 2 bytes
        beq     tx_chksum1
        cmpd    TX_LENGTH(r4),$(1)      # last 1 bytes
        beq     tx_chksum2

        movd    SEND_PTR(r4),r3 # get message address
        movd    BYTE_POS(r4),r5         # get byte position
        movzbd  r3[r5:b],r2             # get message byte
        movb    r2,data_reg(r6)         # output character to IO port
        addqd   $(1),BYTE_POS(r4)       # increment byte position
        addw    r2,TX_CHKSUM(r4)        # perform checksum calculation
        subd    $(1),TX_LENGTH(r4)      # decrement byte count
        br      tx_exit

tx_chksum1::
        movqd   $(1),r5
        movb    TX_CHKSUM(r4)[r5:b],data_reg(r6) # tx checksum hi_byte
        subd    $(1),TX_LENGTH(r4)              # decrement byte count
        br      tx_exit

tx_chksum2::
        movqd   $(0),r5
        movb    TX_CHKSUM(r4)[r5:b],data_reg(r6) # tx checksum lo_byte
        movqb   $(1),int_enable_reg(r6) # disable tx_empty interrupt
        movd    $(9),COM_STATE(r4)              # WAIT for ACK or NAK
        movd    @T2_BUFF,PCB_TIMER(r4)
tx_exit:
        restore [r1,r2,r3,r4,r5,r6,r7]
        movd    tos,r0
        bispsrw $(0x0800)               # enable interrupts
        reti

 ##########################################################################
 # Character transmit routine (ACE xx)=r6 /  r2 = data

put_acex::      # r2 = data, r4 = PCB

        save    [r1,r3,r5,r6,r7]
        movd    PCB_PORT(r4),r1
        addr    ace_addr_bd1,r7
        movd    r7[r1:d],r6
        movzwd  $0x01ff,r5

send_data:      # r2 = data
```

```
        restore [r1,r3,r5,r6,r7]
        ret     0                       # error return

send_data_a:
        movb    r2,data_reg(r6)         # else output character to IO port
        restore [r1,r3,r5,r6,r7]
        ret     0
```

```
#############################################################################
# This file contain the initialization procedure for all the eight ports #
# from ACE board 1.                                                       #
#############################################################################

        .file   "sioini_bd1"

        .globl  ace_11
        .globl  ace_12
        .globl  ace_13
        .globl  ace_14
        .globl  ace_15
        .globl  ace_16
        .globl  ace_17
        .globl  ace_18

        .set    ace_11  ,0xfef800
        .set    ace_12  ,0xfef810
        .set    ace_13  ,0xfef820
        .set    ace_14  ,0xfef830
        .set    ace_15  ,0xfef840
        .set    ace_16  ,0xfef850
        .set    ace_17  ,0xfef860
        .set    ace_18  ,0xfef870

        .set    rx11,   0               # interrupt vector 0  for port 11 Rx
        .set    rx12,   2               # interrupt vector 2  for port 12 Rx
        .set    rx13,   4               # interrupt vector 4  for port 13 Rx
        .set    rx14,   6               # interrupt vector 6  for port 14 Rx
        .set    rx15,   8               # interrupt vector 8  for port 15 Rx
        .set    rx16,   10              # interrupt vector 10 for port 16 Rx
        .set    rx17,   12              # interrupt vector 12 for port 17 Rx
        .set    rx18,   14              # interrupt vector 14 for port 18 Rx
        .set    rx11_vect,(rx11+16)*4   # irRx11 interrupt vector,  ...
        .set    rx12_vect,(rx12+16)*4   # irRx12 interrupt vector,  ...
        .set    rx13_vect,(rx13+16)*4   # irRx13 interrupt vector,  ...
        .set    rx14_vect,(rx14+16)*4   # irRx14 interrupt vector,  ...
        .set    rx15_vect,(rx15+16)*4   # irRx15  interrupt vector,  ...
        .set    rx16_vect,(rx16+16)*4   # irRx16 interrupt vector,  ...
        .set    rx17_vect,(rx17+16)*4   # irRx17 interrupt vector,  ...
        .set    rx18_vect,(rx18+16)*4   # irRx18 interrupt vector,  ...

        .text

# define ace address array.
```

```
        .double 0
        .double ace_11
        .double ace_12
        .double ace_13
        .double ace_14
        .double ace_15
        .double ace_16
        .double ace_17
        .double ace_18

 # Define procedure descriptor for Ace Board 1.

g11xpd: .xpd    get_ace11
g12xpd: .xpd    get_ace12
g13xpd: .xpd    get_ace13
g14xpd: .xpd    get_ace14
g15xpd: .xpd    get_ace15
g16xpd: .xpd    get_ace16
g17xpd: .xpd    get_ace17
g18xpd: .xpd    get_ace18


 # Begin initialization procedure

sioini_bd1::

        bsr     sioini_11
        bsr     sioini_12
        bsr     sioini_13
        bsr     sioini_14
        bsr     sioini_15
        bsr     sioini_16
        bsr     sioini_17
        bsr     sioini_18
        ret     0

 #              Initialize the ns16450 (port 1)
sioini_11:
        save    [r1]
        movd    $(ace_11),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g11xpd,rx11_vect(r1)     # setup RX int. vector
        restore [r1]
        ret     0
```

```
        save    [r1]
        movd    $(ace_12),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g12xpd,rx12_vect(r1)     # setup RX int. vector
        restore [r1]
        ret     0

 #              Initialize the ns16450 (port 3)
sioini_13:
        save    [r1]
        movd    $(ace_13),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g13xpd,rx13_vect(r1)     # setup RX int. vector
        restore [r1]
        ret     0

 #              Initialize the ns16450 (port 4)
sioini_14:
        save    [r1]
        movd    $(ace_14),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g14xpd,rx14_vect(r1)     # setup RX int. vector
        restore [r1]
        ret     0

 #              Initialize the ns16450 (port 5)
sioini_15:
        save    [r1]
        movd    $(ace_15),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g15xpd,rx15_vect(r1)     # setup RX int. vector
        restore [r1]
        ret     0

 #              Initialize the ns16450 (port 6)
sioini_16:
        save    [r1]
        movd    $(ace_16),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g16xpd,rx16_vect(r1)    # setup RX int. vector
```

```
#               Initialize the ns16450 (port 7)
sioini_17:
        save    [r1]
        movd    $(ace_17),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g17xpd,rx17_vect(r1)    # setup RX int. vector
        restore [r1]
        ret     0

#               Initialize the ns16450 (port 8)
sioini_18:
        save    [r1]
        movd    $(ace_18),r1            # ace_3 base address.
        bsr     sioinit
        sprd    intbase,r1              # setup interrupt vectors in IDT:
        movd    g18xpd,rx18_vect(r1)    # setup RX int. vector
        restore [r1]
        ret     0


sioinit:
        movb    $0x80,line_cont_reg(r1) # Set dlab0 = 1 for divisor latch
                                        # access.
## Settings for 9600 baud rate

        movb    $0x0c,divisor_low(r1)   # 9600 with 1.8432mhz clock.
        movb    $0,divisor_high(r1)     # Upper divisor = 0.

# Settings for 300 baud rate

##      movb    $0x80,divisor_low(r1)   # 300 (384) with 1.8432mhz clock.
##      movb    $0x01,divisor_high(r1)  # Upper divisor = 01.

        movqb   $3,line_cont_reg(r1)    # Dlab0 = 0,8 bits,no parity,1 stop
                                        # bit.
        movqb   $(1),int_enable_reg(r1) # Enable ace receive buffer full intrpt.

        movqb   $2,modem_cont_reg(r1)   # Clear out1,out2 and enable rts.
        ret     0
```

```
        .file   "hook1_pxi.asm"

#########################################################################
# This file is part of PXI communication package that must be copied
# to the user directory. It allows the user to define additional
# message checking procedure for all incoming PXI message.
# The standard checks done by PXI communication package on all incoming
# PXI messages is as follows :-
#
#               1. Known TYPE (GEIC board SW7 setting)
#               2. Known GEIC ID (GEIC board SW6 setting)
#               3. Host ONLINE
#               4. Known PIP_ID
#               6. PRIMARY / EXPECTED MESSAGE
#               5. * User define checks (HOOK1_PXI)
#               7. Supported Stream & Function
#
#       Input:  r3 = Message Address
#               r4 = State Buffer address
#               r7 = 0
#
#       Return: r3 = Message address
#               r4 = State Buffer address
#               r7 = 0 ( check success)
#             or r7 not equal zero (check failure)
#########################################################################

#=======================================================================
# declare the USER include file here.
#
#include "vsii_equ.inc"
#
#=======================================================================

        .text
#-----------------------------------------------------------------------
hook1_pxi::     # @@IMPORTANT: DO NOT CHANGE THIS LABEL NAME
#-----------------------------------------------------------------------

# equipment id = 0 - message for geic

        addr    PXB_BUFFER,r4
        movqd   $(0),r7
        cmpb    PXI_MID(r3),$(0)
        beq     hook1_ret
```

```
#=============================================================================
# This section is specific to XPC operation.
# message for re-routing to equipment. Is equipment supported ?
#=============================================================================
# verify if equipment is attached to this geic

        bsr     verify_eqid
        cmpd    r7,$(-1)
        beq     hook1_ret
        movqd   $(0),r7
#=============================================================================
hook1_ret:
        ret     0
```

```
#=============================================================================
# This section is specific to XPC operation.
# message for re-routing to equipment. Is equipment supported ?
#=============================================================================
# verify if equipment is attached to this geic

        bsr     verify_eqid
        cmpd    r7,$(-1)
        beq     hook1_ret
        movqd   $(0),r7
#=============================================================================
hook1_ret:
        ret     0
```

```
        .file   "hook4_pxi.asm"

#include "vsii_equ.inc"

        .text

 ###########################################################################
 # this label is used by pxi mon to call program diagnostic subroutine.
 ###########################################################################
prog_diag::

 #--------------------------------------------------------------------------

xpc_diag:

        movd    $ace_3,r5
        bsr     ed2             # erase complete screen
        bsr     p_header
        movd    $(0x3133),r1    # top - row 8=13
        movd    $(0x3233),r2    # bot - row 23
        bsr     decstbm         # set top bottom margin
        bsr     dectcemi        # set invisible cursor

 ###########################################################################
p_diag::
        movd    $ace_3,r5

        movd    $(0x3133),r1            # row 13
        movd    $(0x3031),r2            # column 1
        bsr     cup                     # set cursor position

        bsr     p_status
        bsr     p_data

finish:
        movd    $(1000),r3
        movd    $QUE_ACE3,r2
        movd    $SCFQPEND,r0
        svc

        cmpd    $(0),r0
        bne     p_diag
        cmpb    $(0x0d),r3
        beq     xpc_diag
        cmpb    $(0x1a),r3
```

```
#*************************************************************************
# Display Header

p_text: .ascii  "*XPC PORT STATUS SCREEN"

p_text1:
        .ascii  "-----------------------"

p_header:
        movd    $(0x3031),r1            # row 1
        movd    $(0x3038),r2            # column 17
        bsr     cup                     # set cursor position
        bsr     decdwl                  # double width
        addr    p_text,r4
        movd    $(23),r7
        bsr     display

        movd    $(0x3032),r1            # row 2
        movd    $(0x3038),r2            # column 17
        bsr     cup                     # set cursor position
        bsr     decdwl                  # double width
        addr    p_text1,r4
        movd    $(23),r7
        bsr     display

        movd    $(0x3034),r1            # row 4
        movd    $(0x3031),r2            # column 17
        bsr     cup                     # set cursor position
        bsr     decswl                  # single_width

        ret     0

#*************************************************************************
# Display PIP OFLINE or ONLINE

ofl:    .ascii  " HOST OFLINE " # HOST STATUS = 0
onl:    .ascii  " HOST ONLINE " # HOST STATUS = 1

string1:
        .ascii  "TYPE: "

string2:
        .ascii  "GEIC_ID: "

string3:
```

```
        .ascii  "BUFFER= "

string4a:
        .ascii  " : "

p_status::
        bsr     decsc                   # save cursor position

 #-------------------------------------------------------------------------
 # display PXI OPERATION TYPE

        movd    $(0x3033),r1            # row 3
        movd    $(0x3136),r2            # column 11
        bsr     cup                     # set cursor position

        addr    string1,r4
        movd    $(6),r7
        bsr     display

        movzbd  @type_switch,r2
        bsr     cnvrt_ascii
        addqd   $(5),r4
        movd    $(3),r7
        bsr     display

 #-------------------------------------------------------------------------
 # display GEIC ID

        movd    $(0x3033),r1            # row 3
        movd    $(0x3238),r2            # column 28
        bsr     cup                     # set cursor position

        addr    string2,r4
        movd    $(9),r7
        bsr     display

        movzbd  @id_switch,r2
        bsr     cnvrt_ascii
        addqd   $(5),r4
        movd    $(3),r7
        bsr     display

 #-------------------------------------------------------------------------
 # display PIP ID
```

```
        addr    string3,r4
        movd    $(8),r7
        bsr     display

        movzwd  @PIPID,r2
        rotw    $(8),r2
        bsr     cnvrt_ascii
        movd    $(8),r7
        bsr     display

#-------------------------------------------------------------------------
# display SECS buffer size

        movd    $(0x3034),r1            # row 4
        movd    $(0x3136),r2            # column 16
        bsr     cup                     # set cursor position

        addr    string4,r4
        movd    $(8),r7
        bsr     display

        movd    $SEC_COUNT,r2
        bsr     cnvrt_ascii
        addqd   $(4),r4
        movd    $(4),r7
        bsr     display

        addr    string4a,r4
        movd    $(3),r7
        bsr     display

        movd    @USE_BUF_COUNT,r2
        bsr     cnvrt_ascii
        addqd   $(4),r4
        movd    $(4),r7
        bsr     display

# display line status
        movd    $(0x3034),r1            # row 4
        movd    $(0x3437),r2            # column 55
        bsr     cup                     # set cursor position
        bsr     sgr7                    # reverse_video

        addr    on1,r4
```

```
        addr    of1,r4

disp_stat:
        movd    $(13),r7
        bsr     display
        bsr     sgr27                   # normal_video
        bsr     decrc                   # restore cursor position
        ret     0

 ##########################################################################

string5:
.ascii "INPUT    EQ CM MID RECV SEND T1   T2   T3   T4   RTY  MODEL  SOFTR

string6:
.ascii "-------- -- -- --- ---- ---- ---- ---- ---- ---- ---- ------ -----

string7:
        .ascii "PORT "

p_data::
        movd    $(0x3036),r1            # row 6
        movd    $(0x3035),r2            # column 4
        bsr     cup                     # set cursor position
        addr    string5,r4
        movd    $(67),r7
        bsr     display

        movd    $(0x3037),r1            # row 6
        movd    $(0x3035),r2            # column 4
        bsr     cup                     # set cursor position
        addr    string6,r4
        movd    $(67),r7
        bsr     display

        addr    PCB_BUFFER,r6
        movd    $(PCB_COUNT),r7

 next_pcb:
        save    [r7]
        movd    $(0x0d),r2
        bsr     put_ace3
        movd    $(0x0a),r2
        bsr     put_ace3
        movd    $(0x0d),r2
```

```
bsr     cuf

addr    string7,r4
movd    $(5),r7
bsr     display

movd    PCB_PORT(r6),r2
bsr     cnvrt_ascii
addqd   $(6),r4
movd    $(2),r7
bsr     display

movd    $(0x3a),r2      # :
bsr     put_ace3
movd    $(0x20),r2      # sp
bsr     put_ace3

movd    EQP_STATE(r6),r2
bsr     cnvrt_ascii
addqd   $(6),r4
movd    $(2),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    COM_STATE(r6),r2
bsr     cnvrt_ascii
addqd   $(6),r4
movd    $(2),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movzbd  PCB_MID(r6),r2
bsr     cnvrt_ascii
addqd   $(5),r4
movd    $(3),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    MSG_RX_COUNT(r6),r2
```

```
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    MSG_TX_COUNT(r6),r2
bsr     cnvrt_ascii
addqd   $(4),r4
movd    $(4),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    T1_COUNT(r6),r2
bsr     cnvrt_ascii
addqd   $(4),r4
movd    $(4),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    T2_COUNT(r6),r2
bsr     cnvrt_ascii
addqd   $(4),r4
movd    $(4),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    T3_COUNT(r6),r2
bsr     cnvrt_ascii
addqd   $(4),r4
movd    $(4),r7
bsr     display

movd    $(0x20),r2
bsr     put_ace3

movd    T4_COUNT(r6),r2
bsr     cnvrt_ascii
addqd   $(4),r4
movd    $(4),r7
```

```
        .file   "pxi_port.asm"

#include "pxi_equ.inc"

.text

 ##########################################################################
 # This task perform the following function.
 #      1. Use VRTX GETC function to read a character from ACE_1 port.
 #      2. Save the character in R3(b)
 #      3. Set R3 bit-31 because if the character received is a
 #         null(00) character, it cannot be posted.
 #
 #      Note: The character posted is used by PXI_BXFER task.
 ##########################################################################

pxi_port::

 # Wait for Character from ACE_1 port. Character is returned in r2[b].

        movd    $SCFGETC,r0
        svc
        cmpd    $(0),r0                 # getc error ?
        bne     pxi_port                # yes

 # if host is initializing, discard character.

        tbitd   $(31),@CON_STATUS       # host ofline initialization
        bfs     pxi_port                # in progress ? - yes

 # transfer character(r2[b]) to r3. for posting.

        movzbd  r2,r3                   # save char. in R3
        sbitd   $(31),r3                # set bit-31 of R3
        movd    $(QUE_PXI_PORT),r2      # post character
        movd    $SCFQPOST,r0
        svc

        br      pxi_port                # get next char.
```

```
        .file "pxi_bxfer.asm"
#############################################################################
# The block transfer (BXFER) task is the communication package used
# to Receive and Transmit PXI messages.
#
# This program follows roughly the flow as defined in the 1987 SEMI-
# STANDARDS manual page 5 BLOCK TRANSFER PROTOCOL flow diagram.
#############################################################################

#include "pxi_equ.inc"

#-----------------------------------------------------------------------
# Declare some common macro used
#-----------------------------------------------------------------------
# TRANSMIT a byte to ACE_1 port.

        .macro  tx_byte
        movd    $SCFPUTC,r0             # Transmit byte through
        svc                             # GEIC ACE-1 port.
        .endm                           #

#-----------------------------------------------------------------------
# RECEIVE a byte from ACE_1 port. r3 supplies the timeout value.

        .macro  rx_byte
        movd    $QUE_PXI_PORT,r2        # Read byte for Queue
        movd    $SCFQPEND,r0            #
        svc                             #
        .endm                           #

#-----------------------------------------------------------------------

        .text

#############################################################################
#     IDLE LOOP / IDLE LOOP / IDLE LOOP / IDLE LOOP / IDLE / LOOP
#############################################################################

pxi_bxfer::
idle:

#-----------------------------------------------------------------------
# Mark BXFER task is idle. The PXI_MSGTX task uses this mail box
# to determine whether the block transfer task is 'busy'.
#-----------------------------------------------------------------------
        sbith   $(7),@msg_port          # Turn LED 8 'OFF'
```

```
        movd    $SCFPOST,r0             #
        svc                             #

#-------------------------------------------------------------------------
# wait for ENQ (0x05) from PXI_PORT  task (incoming line request)
# or       SND (0xff) from PXI_MSGTX task (outgoing line request)
#-------------------------------------------------------------------------
idle_a:
        movqd   $(0),r3                 # Indefinite wait on queue
        movd    $QUE_PXI_PORT,r2        #
        movd    $SCFQREND,r0            #
        svc                             #
        cbitb   $(7),@msg_port          # Turn LED 8 'ON'
        cmpd    r0,$(0)                 # Que error ?
        bne     idle                    # yes.

#-------------------------------------------------------------------------
        cmpb    r3,$ENQ                 # Incoming request for line ?
        beq     incoming_message        # Yes.
                                        #
        cmpb    r3,$SND                 # Outgoing request for line ?
        beq     outgoing_message        # yes.
                                        #
        br      idle                    # Disregard input (garbage).

#########################################################################
#    LINE CONTROL / LINE CONTROL / LINE CONTROL / LINE CONTROL
#########################################################################
incoming_message:

# this section is used to inform PXI_MSGTX task that BXFER is busy.
# PXI_MGSTX task must not send a request for outgoing line.

        addr    MBX_PXI_BXFER_STATUS,r1 # Mark BXFER task 'busy'.
        movd    $(10),r2                # 10ms timeout
        movd    $SCFPEND,r0             #
        svc                             #
        cmpd    r0,$(0)                 # Error/timeout
        bne     idle                    # Unable to mark BXFER busy.

#-------------------------------------------------------------------------
        bsr     get_RCV_BUF             # Request for buffer space.
        cmpd    r3,$(-1)                # r3 = buffer address or error.
        beq     idle                    # Buffer full. Request denied.
                                        #
```

```
        br      receive                 # Begin RECEIVE sequence.

 ************************************************************************
outgoing_message:
        movd    $PXI_RETRY,r4           # Set retries count.

send_enq:
        movb    $ENQ,r2                 # Send request for line.
        tx_byte                         #

wait_eot:
        movd    $PXI_T2,r3              # Set Protocol timeout value.
        rx_byte                         # <LISTEN> for EOT.
        cmpd    r0,$(0)                 # ER_TMO ?
        bne     protocol_timeout        # Timeout has occured.
                                        #
        cmpb    r3,$EOT                 # Line granted ?
        bne     contention              # Receive other than EOT.
        br      send                    # Begin SEND sequence.

 ************************************************************************
 # RECEIVE / RECEIVE / RECEIVE / RECEIVE / RECEIVE / RECEIVE / RECEIVE
 ************************************************************************
receive:
        movd    @PXI_RECD_PTR,r7        # Load buffer address
        movqd   $(0),r6                 # Set byte position.
                                        #
        movd    $PXI_T2,r3              # Set Protocol timeout value.
        rx_byte                         # <LISTEN> Upper LEN_WORD.
        cmpd    r0,$(0)                 # ER_TMO ?
        bne     inter_char_timeout      # Timeout has occured.
                                        #
        movb    r3,LEN_WORD(r7)         # Save Upper LEN_WORD.
                                        #
        movd    $PXI_T2,r3              # Set Protocol timeout value.
        rx_byte                         # <LISTEN> Lower LEN_WORD.
        cmpd    r0,$(0)                 # ER_TMO ?
        bne     inter_char_timeout      # Timeout has occured.
                                        #
        movb    r3,LEN_WORD+1(r7)       # Save Lower LEN_WORD.
                                        #
        movzwd  LEN_WORD(r7),r5         # Set length word counter.
        rotw    $(8),r5                 # Swap byte.
        cmpw    r5,$(0)                 # Verify if LEN_WORD = 0
        bne     receive_char            # NO.
```

```
# not be able to deallocate the buffer and decrement DIAG_USE
# counter because the first four byte of the buffer is still zero(0).

        movd    $(-1),LEN_WORD(r7)      # Mark block for deallocation

#-------------------------------------------------------------------------
# Zero State Polling Procedure. (LEN_WORD = 0)
#-------------------------------------------------------------------------
        movb    $ACK,r2                 # Presence poll ack.
        tx_byte                         #

# verify if GEIC has just startup. If so, inform Host about it.

        cmpd    @HOST_STATUS,$(1)       # GEIC already online ?
        beq     abort                   # Yes. RECEIVE completed.
        bsr     s151f1                  # VSII Startup Request.
        br      abort                   # RECEIVE completed.

#-------------------------------------------------------------------------
receive_char:
        movd    $PXI_T1,r3              # Set Intercharacter timeout value.
        rx_byte                         # <LISTEN> for next character
        cmpd    r0,$(0)                 # ER_TMO ?
        bne     inter_char_timeout      # Timeout has occured.
                                        #
        addqd   $(1),r6                 # Get next byte position.
        movb    r3,1(r7)[r6:b]          # Save received byte.
        acbw    $(-1),r5,receive_char_a # <DONE> ?
        br      message_received        # YES.

receive_char_a:
        cmpd    r6,$(PXI_MAX)           # End of current buffer ?
        blo     receive_char            # NO
                                        #
        bsr     get_RCV_BUF             # Request for additional buffer space.
        cmpd    r3,$(-1)                # r3 = buffer address or error.
        beq     completion              # Buffer full. Abort RECEIVE sequence.
                                        #
        movd    r3,NBLOCK_CHAIN(r7)     # Link new buffer address into chain.
        movd    r3,r7                   # Load new buffer address.
        movqd   $(0),r6                 # Reset byte position.
        br      receive_char            # Resume RECEIVE sequence.

#-------------------------------------------------------------------------
message_received:
```

```
        svc                             #
        cmpd    r0,$(0)                 # Post successfull ?
        bne     completion              # Error - Unable to post.

        addqd   $(1),@RCV_COUNTER       # Increment RX message pending count.
        addqd   $(1),@DIAG_RCV          # Increment total RX message count.
        movqd   $(0),@PXI_RECD_PTR      # Reset pointer.
                                        #
        movb    $ACK,r2                 # ACKnowledge message received.
        tx_byte                         #
                                        #
        br      idle                    # BXFER task 'idle'

 ########################################################################
 #   COMPLETION / COMPLETION / COMPLETION / COMPLETION / COMPLETION
 ########################################################################

 # Receiver continue receiving until sender is finish sending before
 # replying with a NAK byte.

completion:
        movd    $PXI_T1,r3              # Set intercharacter timeout value.
        rx_byte                         # <LISTEN> for next character.
        cmpd    r0,$(0)                 # ER_TMO ?
        beq     completion              # NO. Wait for next char.

inter_char_timeout:
        addqd   $(1),@DIAG_RNK          #
        movb    $NAK,r2                 # Send NAK to Host
        tx_byte                         #

abort:  movd    @PXI_RECD_PTR,r3        # Load buffer address.
        movqd   $(0),@PXI_RECD_PTR      # Reset pointer.
        bsr     pxi_discard             # Deallocate buffer space.
        br      idle                    # BXFER task 'idle'.

 ########################################################################
 #    SEND / SEND / SEND / SEND / SEND / SEND / SEND / SEND / SEND
 ########################################################################

send:
        movd    @PXI_SEND_PTR,r7        # Load 1st block message address.
        movzwd  LEN_WORD(r7),r5         # Load message length word.
        rotw    $(8),r5                 # r5 = total message length.
        movqd   $(0),r6                 # r6 = current block byte position.
```

```
        movb    LEN_WORD(r7),r2         # Send Upper LEN_WORD.
        tx_byte                         #
        movb    LEN_WORD+1(r7),r2       # Send lower LEN_WORD.
        tx_byte                         #

 # send N byte. ********************************************************
n_byte:
        movd    @INT_BUFF,r2            # Set intercharacter delay
        movd    $SCFTDELAY,r0           # because host cannot respond
        svc                             # to zero delay transmit.

        addqd   $(1),r3                 # Increment byte counter.
        addqb   $(1),r6                 # Increment byte position.
        movb    1(r7)[r6:b],r2          # Get data byte.
        tx_byte                         # Send byte.
        acbw    $(-1),r5,n_byte_a       # DONE ?
        br      message_sent            # YES.

n_byte_a:
        cmpd    r3,$(INTER_BLK_SIZE)    # Interblock limit reached ?
        bne     n_byte_b                # No

 # Interblock limit reached. Wait for Hosk ACK before proceeding.

        movd    $PXI_T2,r3              # Set protocol timeout value.
        rx_byte                         # <LISTEN> for protocol ACK
        cmpd    r0,$(0)                 # ER_TMO ?
        bne     ack_timeout             # Timeout has occured.

        cmpb    r3,$ACK                 # Host received block correctly ?
        bne     nak_retry               # No. Host encounter error.

        movqd   $(0),r3                 # Block receive correct.
                                        # Reset byte counter.

n_byte_b:
        cmpd    r6,$(PXI_MAX)           # End of current buffer ?
        blo     n_byte                  # No. Continue to send.
        movd    NBLOCK_CHAIN(r7),r7     # Get next buffer address
        movqd   $(0),r6                 # from Chain pointer and
        br      n_byte                  # continue to send.

 ######################################################################
 # wait for acknowledgement (completion)
```

```
        rx_byte                         # <LISTEN> for protocol ACK.
        cmpd    r0,$(0)                 # ER_TMO ?
        bne     ack_timeout             # Timeout has occured.

        cmpb    r3,$ACK                 # message sent success ?
        bne     nak_retry               # NO - Retry.

        movw    $(0x8000),r2            # Send SUCCESS (r2 =00)

#-------------------------------------------------------------------
# Report to MSGTX task the result of the Send Request.
# r2[b] =  0 - Success,
#          -1 - Protocol timeout. No respond from Host.
#          -2 - Timeout waiting for ACK
#          -3 - Host respond with NAK
#-------------------------------------------------------------------
block_sent:
        movd    $(INTER_CHAR_DELAY),@INT_BUFF

        addr    MBX_PXI_SEND_STATUS,r1  # return send status to requestor
        movd    $SCFPOST,r0             # SC_POST function code
        svc
        br      idle

#-------------------------------------------------------------------
# timeout/error encountered during message send
#-------------------------------------------------------------------

contention:     # Send ENQ but receive other than EOT

        movd    $(-4),r2                # Error code r2 = -4
        cmpb    r3,$ENQ                 # Receive other than EOT or ENQ.
        bne     retry_count             # Sequencing error. Retry.

 # Host respond with ENQ instead of EOT. After retry count is exhausted,
 # assume that Host is in STARTUP mode. HOST will be marked as OFLINE .

        addqd   $(1),@DIAG_CONTN        # contention counter
        acbd    $(-1),r4,send_enq       # Retry Send.
        movd    $(-1),r2                # Error code r2 = -1 (OFLINE)
        br      block_sent              # Retry exhausted.

#-------------------------------------------------------------------
protocol_timeout:
        movd    $(-1),r2                # Error code r2 = -1
```

```
        br      retry_count             # RETRY.
nak_retry:
        addqd   $(1),@DIAG_TNK          # Increment NAK counter
        movd    $(-3),r2                # Error code r2 = -3

 #-------------------------------------------------------------------------
retry_count:
        addd    $(INTER_CHAR_RANGE),@INT_BUFF # increase inter_char_delay time
        addqd   $(1),@DIAG_RTY          # Increment retry counter
        acbb    $(-1),r4,send_enq       # Next retry.
        br      block_sent              # Retry Exhausted.
```

```
        .file   "pxi_msgrx.asm"

#include "pxi_equ.inc"

        .text
pxi_msgrx::

 ##########################################################################
 # this section waits indefinitely for any incoming pxi messages
 ##########################################################################

msgrx_idle:
        sbitb   $(6),@msg_port                  # Turn LED 7 'OFF'
        movqd   $(0),r3                         #
        movd    $QUE_PXI_MSGRX,r2               #
        movd    $SCFQPEND,r0                    #
        svc                                     #
        cbitb   $(6),@msg_port                  # Turn LED 7 'ON'
        cmpd    r0,$(0)                         # Queue error ?
        bne     msgrx_idle                      # Yes. Disregard.

 # Perform some basic checking to verify message integrity.

        cmpd    @RCV_COUNTER,$(0)               # Counter = 0 ?
        beq     msgrx_idle                      # Yes. Disregard.

        bsr     verify_pxi_addr                 # Verify message address.
        cmpd    r3,$(-1)                        # Error ?
        beq     msgrx_idle                      # Yes.

  #-------------------------------------------------------------------------
  # received pxi message
  #-------------------------------------------------------------------------
        subd    $(1),@RCV_COUNTER               # Decrement count.
        movqd   $(0),@PXI_PRIMARY               # Clear pointer.
        movw    PXI_AID(r3),@APPID              # Save APPLICATION ID.
        movd    r3,@SAVE_MSGRX                  # Save Message address.
        addr    PXB_BUFFER,r4                   # Host PCB address

  #-------------------------------------------------------------------------
  # verify operation type.
  #-------------------------------------------------------------------------

        cmpb    PXI_TYP(r3),@type_switch        # Supported TYPE ?
        bne     unsupported_type                # No. Error.
```

```
#-------------------------------------------------------------------------

        cmpb    PXI_GID(r3),@id_switch          # Supported GID ?
        bne     illegal_gid                     # No. Error or pip_startup

#-------------------------------------------------------------------------
# verify if geic has just startup. Host status = ofline
#-------------------------------------------------------------------------

        cmpd    @HOST_STATUS,$(0)               # GEIC Startup Mode ?
        beq     geic_startup                    # Yes.

#-------------------------------------------------------------------------
# verify if pxi message is from the correct pip task.
#-------------------------------------------------------------------------

verify_pipid:
        cmpw    PXI_PID(r3),@PIPID              # Supported PID ?
        bne     unsupported_pipid               # No. Error - PID has chan

#-------------------------------------------------------------------------
# verify if message is a primary message.
#-------------------------------------------------------------------------
pxi_epb:
        tbitb   $(0),PXI_FNC(r3)                # Primary Message ?
        bfs     pxi_sf_support                  # Yes

#-------------------------------------------------------------------------
# SECONDARY MESSAGE. Verify if message is expected.
#-------------------------------------------------------------------------
        addr    EPB_BUFFER,r0                   # Load Expected Block addr
        movd    $EPB_COUNT,r7                   # Load block count.

pxi_epb_a:
        cmpd    EPB_TIMER(r0),$(0)              # Empty EPB block ?
        beq     pxi_epb_b                       # Yes - get next block.
        cmpd    PXI_SYS(r3),EPB_SYSBYT(r0)      # System byte same ?
        beq     pxi_epb_c                       # Yes - expected_block

pxi_epb_b:
        addd    $(EPB_SIZE),r0                  # Next Block location
        acbd    $(-1),r7,pxi_epb_a              # Last ?
        br      rcv_deallocate                  # Yes. Unexpected message.

# EXPECTED Message.
```

```
        movqd   $(0),EPB_TIMER(r0)          # clear EPB block - timer.
        subd    $(1),@EPB_COUNTER           # decrement counter

 #-------------------------------------------------------------------------
 # PRIMARY or EXPECTED SECONDARY MESSAGE.
 #-------------------------------------------------------------------------
pxi_sf_support:
 ##########################################################################
        bsr     hook1_pxi       # USER DEFINE CHECKS
        cmpd    r7,$(-1)        # Fail user define checks ?
        beq     unsupported_mchid       # error

 ##########################################################################
verify_sf::
        movqd   $(0),@PXI_STREAM_FLAG   # reset stream marker
        addr    message_table,r7

verify_sf_a:
        cmpw    0(r7),$(-1)             # end of table
        beq     unsupported_message

        cmpb    0(r7),PXI_STM(r3)
        bne     next_sf
        movqd   $(1),@PXI_STREAM_FLAG   # mark stream marker
        cmpb    1(r7),PXI_FNC(r3)
        beq     valid_sf

next_sf:
        addd    $(TAB_SIZE),r7
        br      verify_sf_a

valid_sf::
        movd    PXI_STATE(r4),r5        # Load Equipment State Value
        movd    2(r7)[r5:d],r6          # r6 = subroutine address.

 #-------------------------------------------------------------------------
 # This section passes the message to the user service routine
 # r3 = message address
 # r4 = PCB address
 # PXI_PRIMARY = Primary message or 0x0000
 #
 # Note : The received message will be deallocated when it returns
 #        from the User Service Routine (U.S.R).
 #-------------------------------------------------------------------------
        jsr     r6                      # Goto U.S.R.
```

```
 # this section is used to report unsupported messages

unsupported_mchid:
        bsr     s159f65
        movd    $(-1),r7
        br      rcv_deallocate

unsupported_message:
        cmpd    @PXI_STREAM_FLAG,$(0)
        bne     unsupported_func

unsupported_stream:
        bsr     s159f3
        movd    $(-2),r7
        br      rcv_deallocate

unsupported_func:
        bsr     s159f5
        movd    $(-3),r7

 #-------------------------------------------------------------------------
rcv_deallocate:
        movd    @SAVE_MSGRX,r3          # restore current message address
        movqd   $(0),@SAVE_MSGRX        # reset pointer.
        movqd   $(0),@APPID             # reset current application id
        bsr     pxi_discard             # discard current message
        cmpd    @PXI_PRIMARY,$(0)       # pointer = primary addr.
        beq     msgrx_idle              # RECEIVE COMPLETED.

 # deallocate primary message

        movd    @PXI_PRIMARY,r3         # Load primary message addr
        movqd   $(0),PXI_PRIMARY        # Clear pointer
        bsr     pxi_discard             # Discard message.
        br      msgrx_idle              # RECEIVE COMPLETED.

 ##########################################################################
 # END OF MAIN PROGRAM
 ##########################################################################

 #-------------------------------------------------------------------------
 # this section is used when the GEIC is in the STARTUP MODE due to :-
 # 1. GEIC reset or Power Up.
 # 2. GEIC that Host is "OFLINE" after protocol timeout has occured.
 #
```

```
geic_startup:
        cmpb    PXI_STM(r3),$(151)      # Stream 151
        bne     geic_startup_a
        cmpb    PXI_FNC(r3),$(2)        # Function 2
        beq     pxi_epb

geic_startup_a:
        bsr     s151f1
        br      rcv_deallocate

 #-------------------------------------------------------------------------
 # this section is used when :-
 # 1. PIP just startup and does not know the GEIC ID. (GID = 0)
 # 2. The port has been switched

illegal_gid:
        cmpb    PXI_GID(r3),$(0)        # GID = 0
        bne     unsupported_gid         # no

 # if GID = 0, then verify if message = s151f1

        cmpb    PXI_STM(r3),$(151)      # Stream 151
        bne     unsupported_gid
        cmpb    PXI_FNC(r3),$(1)        # Function 1
        bne     unsupported_gid

 # yes. GID = 0 and message = s151f1

        cmpd    @HOST_STATUS,$(1)       # originally online
        beq     verify_pipid            # pid changed

        bsr     pip_startup
        br      rcv_deallocate

 #-------------------------------------------------------------------------
 # this section is used in the event when
 #      1. the port has been switched ( Wrong PIPID,GID)
 #      2. Wrong operation type task used

unsupported_type:
unsupported_gid:
unsupported_pipid:

        cmpb    PXI_STM(r3),$(159)      # Stream 159
        bne     unsupported_ID
        cmpb    PXI_FNC(r3),$(1)        # Function 1
```

```
# Receive S159F1 (Unsuppported ID) from Host.
# Suspend communication (Ofline) and wait for Host Startup.

        movd    @SAVE_MSGRX,r3
        movqd   $(0),@SAVE_MSGRX
        bsr     pxi_discard             # Discard original message.
        bsr     ofline                  # Reset memory buffer.
        br      msgrx_idle              # Return IDLE.

# Receive message other than S159F1 from Host with illegal ID.
# Send S159F1 to Host.
# Suspend communication (Ofline) and wait for Host Startup.

unsupported_ID:
        movd    @SAVE_MSGRX,r0          # Restore current message address.
        movb    $(159),r1               # Stream 159.
        movqb   $(1),r2                 # Function 1.
        movqd   $(0),SYSBYT_SEC(r3)     # Clear system bytes.
        movqd   $(0),NBLOCK_CHAIN(r3)   # Clear chain pointer.
        bsr     pxi_bldhdr              # Build header w/o get_addr.
        cmpd    r3,$(-1)                # Buffer full ?
        bne     unsupported             # NO

 # ERROR - SND_BUFFER Full.

        bsr     backup_buffer           # Use Backup buffer
        bsr     pxi_bldhdr_x            # to build the message

unsupported:
        movw    PXI_AID(r0),PXI_AID(r3) # Return original application ID.
        movw    PXI_PID(r0),PXI_PID(r3) #

        addr    PXI_AID(r0),r2          # Tag original message header
        movb    $(0x21),r1              # in the S159F1 message.
        movd    $(0x0e),r7              #
        bsr     pxi_blditm              #

        save    [r3]
        movd    @SAVE_MSGRX,r3
        movqd   $(0),@SAVE_MSGRX
        bsr     pxi_discard             # Discard original message.
        bsr     ofline                  # Reset memory buffer.
        restore [r3]

        bsr     pxi_sndmsg              # Send s159f1 to host.
```

```
        .file   "hook2_vsii.asm" # Same as (hook2_pxi.asm)
#-------------------------------------------------------------------------
# Define the message table variables

        .globl  STATE_COUNT
        .set    STATE_COUNT,1   # Number of state for User action

        .globl  TAB_SIZE        # s/f table size
        .set    TAB_SIZE,2+4*STATE_COUNT

        .globl  ERR_TAB_SIZE    # error s/f table size
        .set    ERR_TAB_SIZE,2+4*STATE_COUNT

        .text

message_table::

 #-------------------------------------------------------------------------
 # Streams and Function supported by USER.
 #-------------------------------------------------------------------------

 # SECS messages +++++++++++++++++++++++++++++++

        .byte   2               # stream 2
        .byte   16              # function 25
        .double sec_pxi_send    # Loopback diagnostic request   [SEC]

        .byte   5               # stream 5
        .byte   1               # function 1
        .double s5f1            # alarm request [sec_mgr.asm]   [SEC]

        .byte   6               # stream 6
        .byte   3               # function 3
        .double s5f1            # Discrete val data send        [SEC]

        .byte   1               # stream 1
        .byte   6               # function 6
        .double sec_pxi_send    # Formatted status data         [SEC]


 # PXI messages +++++++++++++++++++++++++++++++

        .byte   2               # stream 2
        .byte   15              # function 25
```

```
#-----------------------------------------------------------------------
# Message supported by PXI - Standard PXI system message.
#-----------------------------------------------------------------------

        .byte   151             # stream 151
        .byte   1               # function 1
        .double pip_startup     # startup message               [PXI]

        .byte   151             # stream 151
        .byte   2               # function 2
        .double startup_reply   # startup reply message         [PXI]

        .byte   1               # stream 1
        .byte   1               # function 1
        .double s1f1            # Are you there request         [SEC]

        .byte   1               # stream 1
        .byte   2               # function 2
        .double equip_online    # On Line Data                  [SEC]

#-----------------------------------------------------------------------
#       Date and Time message
#-----------------------------------------------------------------------

        .byte   2               # stream 2
        .byte   17              # function 26
        .double s2f17           # date and time request         [SEC]

        .byte   152             # stream 152
        .byte   18              # function 25
        .double xpc_date_time   # Date and time data            [PXI]

#-----------------------------------------------------------------------
#       Loopback Diagnostic Message
#-----------------------------------------------------------------------

        .byte   152             # stream 152
        .byte   25              # function 25
        .double xpc_diag_req    # Loopback diagnostic request   [PXI]

        .byte   152             # stream 152
        .byte   26              # function 26
        .double xpc_diag_dat    # Loopback diagnostic data      [PXI]

        .byte   2               # stream 2
```

```
        .byte   2               # stream 2
        .byte   26              # function 26
        .double sec_pxi_send    # Loopback diagnostic data      [SEC]

#-------------------------------------------------------------------------
#       System Error Messages (HOST -> XPC)
#-------------------------------------------------------------------------

        .byte   159             # stream 159
        .byte   167             # function 167
        .double duplicate_eqid  # Duplicate EQID (Host level)   [PXI]

        .byte   159             # stream 159
        .byte   1               # function 1
        .double ofline          # Unsupported PIPID,GID,TYPE    [PXI]

        .byte   159             # stream 159
        .byte   3               # function 3
        .double unsupported     # Unsupported Stream            [PXI]

        .byte   159             # stream 159
        .byte   5               # function 5
        .double unsupported     # Unsupported Function          [PXI]

        .byte   159             # stream 159
        .byte   65              # function 65
        .double unsupported     # Unsupported Equipment         [PXI]

#-------------------------------------------------------------------------
#       Equipment Error Messages (EQUIP -> XPC)
#-------------------------------------------------------------------------

        .byte   9               # stream 9
        .byte   1               # function 1
        .double dummy           # Unrecognised Device ID        [SEC]

        .byte   1               # stream 9
        .byte   65              # function 1
        .double dummy           # Unrecognised Device ID        [SEC]

        .byte   3               # stream 9
        .byte   65              # function 1
        .double dummy           # Unrecognised Device ID        [SEC]

        .byte   64              # stream 9
```

```
        .byte   9               # stream 9
        .byte   3               # function 3
        .double dummy           # Unrecognised Stream           [SEC]

        .byte   9               # stream 9
        .byte   5               # function 5
        .double dummy           # Unrecognised Function         [SEC]

#-------------------------------------------------------------------------
# End of message support table
#-------------------------------------------------------------------------
        .word   -1

#########################################################################
# this table is the list of streams and functions that requires action
# when there is a sent failure.

error_table::
        .byte   152             # stream 152
        .byte   17              # function 17
        .double pxi_resend      # Formatted Variable Send       [PXI]

        .word   -1              # end of error table

#########################################################################
dummy::
        ret     0
#########################################################################
```

```
        .file "sec_mgr.asm"

#include "vsii_equ.inc"

        .text

s5f1::
        bsr     sec_pxi_send    #Send s155f1 => host
        save    [r0,r1,r2,r3]
        movd    r3,r0           #Transfer primary message address r0
        bsr     sec_rephdr      #Build new header and plus stream 1
        cmpd    r3,$(-1)        #Check error
        beq     error_sec
        movd    $(0x21),r1      #format 10
        addr    ACKC5,r2
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     error_sec
        bsr     sec_sndmsg      #Send s5f2

        restore [r0,r1,r2,r3]


        ret     0

ACKC5:  .double 0

s6f3::
        bsr     sec_pxi_send    #Send s156f3 => host
        save    [r0,r1,r2,r3]
        movd    r3,r0           #Transfer primary message address r0
        bsr     sec_rephdr      #Build new header and plus stream 1
        cmpd    r3,$(-1)        #Check error
        beq     error_sec
        movd    $(0x21),r1      #format 10
        addr    ACKC6,r2
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     error_sec
        bsr     sec_sndmsg      #Send s6f4
        restore [r0,r1,r2,r3]

        ret     0

ACKC6:  .double 0
```

```
        save    [r0,r1,r2,r3,r7]
        movb    $(1),r1
        movb    $(5),r2
        movb    $(1),r7         #Need reply
        bsr     sec_bldhdr
        cmpd    r3,$(-1)
        beq     error_sec

        movd    $(0x21),r1      #format 10
        addr    SFCD,r2         #request everything
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     error_sec
        bsr     sec_sndmsg      #Send s1f5

        restore [r0,r1,r2,r3,r7]
        ret     0

SFCD:
        .double 03

 ####simulate
s1f5::
        save    [r0,r1,r2,r3,r4,r7]
        movb    $(1),r1
        movb    $(5),r2
        movb    $(1),r7         #Need reply
        bsr     sec_bldhdr
        cmpd    $(-1),r3
        beq     bug_sec

        movd    $(7),r1                 #list main
        bsr     sec_bldlst
        cmpd    $(-1),r3
        beq     bug_sec
 #1##############################################
        movd    $(2),r1                 #list sub
        bsr     sec_bldlst
        cmpd    $(-1),r3
        beq     bug_sec

        movd    $(0xB1),r1      #format 54
        addr    ECID1,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
```

```
        movd    $(0x65),r1      #format 31
        addr    ECV1,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
#64##################################################
        movd    $(2),r1                 #list sub
        bsr     sec_bldlst
        cmpd    $(-1),r3
        beq     bug_sec

        movd    $(0xB1),r1      #format 54
        addr    ECID64,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
        movd    $(0x65),r1      #format 31
        addr    ECV1,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
#65##################################################
        movd    $(2),r1                 #list sub
        bsr     sec_bldlst
        cmpd    $(-1),r3
        beq     bug_sec

        movd    $(0xB1),r1      #format 54
        addr    ECID65,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
        movd    $(0x65),r1      #format 31
        addr    ECV65,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
#66##################################################
        movd    $(2),r1                 #list sub
        bsr     sec_bldlst
```

```
#97################################################
        movd    $(2),r1                 #list sub
        bsr     sec_bldlst
        cmpd    $(-1),r3
        beq     bug_sec

        movd    $(0xB1),r1      #format 54
        addr    ECID97,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
        movd    $(0x65),r1      #format 31
        addr    ECV96,r2
        movb    $(1),r7         #Size of data
        bsr     sec_blditm      #Build item
        cmpd    r3,$(-1)
        beq     bug_sec
s2f15sn:
        bsr     sec_sndmsg      #Send s2f15

        restore [r0,r1,r2,r3,r4,r7]
        bsr     s51f5
        ret     0
bug_sec:
        restore [r0,r1,r2,r3,r4,r7]
        ret     0
ECID1:
        .double 01
ECID64:
        .double 64
ECID65:
        .double 65
ECID66:
        .double 66
ECID69:
        .double 69
ECID96:
        .double 96
ECID97:
        .double 97
ECV1:
        .double 45
ECV65:
        .double 20
```

```
        .double 0

error_sec::
        ret     0

 ##############################################################################
```

```
--
-- MODIFICATION HISTORY:
--
--          ORIGINAL VERSION, 24 - SEPTEMBER -1991
--          Version 0.1
--          Nimit [edit file name.dat,string size line 89] JAN,11,91
--

with TEXT_IO; use TEXT_IO;
with STARLET;
with CONDITION_HANDLING; use CONDITION_HANDLING;
with TASKING_SERVICES; use TASKING_SERVICES;
with CURRENT_EXCEPTION;
with RTL; use RTL;
with INDEXED_IO;
with SYSTEM; use SYSTEM;
with PVCU; use PVCU;
with FORIN;

pragma ELABORATE ( PVCU );

procedure MPIP is

    pragma TIME_SLICE ( 0.06 );

    type ROUTE_TABLE_RECORD is record
       EQ_ID                : SHORT_INTEGER := 0;
       PORT_ID              : SHORT_INTEGER := 0;
       INDEX                : SHORT_INTEGER := 0;
    end record;

    type MSG_TYPE is ( ZERO_POLL, NORMAL, PRESENCE_POLL );

    type PORT_CONTROL_RECORD is record
        PORT_NAME       : STRING(1..10) := (others => ' ');
        PORT_NAME_LEN   : INTEGER := 0;
        PORT_CHANNEL    : STARLET.CHANNEL_TYPE := 0;
        PUT_PORT_STATUS : VCU_STATUS_TYPE := NONE;
        GET_PORT_STATUS : VCU_STATUS_TYPE := NONE;
        STARTUP_STATUS  : VCU_STATUS_TYPE := NONE;
        PRESENCE_POLLING: SHORT_SHORT_INTEGER := 0;
        STARTED         : BOOLEAN := FALSE;
        MSG_SENT        : BOOLEAN := FALSE;
        ZERO_POLLING    : BOOLEAN := FALSE;
        XPC_ID          : SYSTEM.UNSIGNED_BYTE := 0;
        OLD_XPC_ID      : SYSTEM.UNSIGNED_BYTE := 0;
```

```
type PORT_ARRAY is array ( 1..40 ) of PORT_CONTROL_RECORD;

PORT_DATA            : PORT_ARRAY;
GET_MBOX_STATUS      : VCU_STATUS_TYPE := NONE;

MAX_PRESENCE_COUNT   : constant := 25;

MAX_NO_APPL              : constant := 40;
APPL_PID                 : array (INTEGER range 1..MAX_NO_APPL)
                           of UNSIGNED_LONGWORD := (others => 0 );
NO_OF_APPL               : INTEGER := 0;
MPIP_MBOX_PTR            : MBOX_ACC_TYPE;
MPIP_ID                  : SYSTEM.UNSIGNED_WORD := 0;

MACHINE_TYPE             : STRING(1..2) := "  ";-- Machine type to support.
MACHINE_TYPE_LENGTH      : INTEGER := 0;

NO_OF_PORTS              : STRING(1..2);        -- No of ports to support.
NO_OF_PORTS_LENGTH       : INTEGER := 0;
TOTAL_PORTS              : INTEGER := 0;

MPIP_MAIL_BOX_NAME       : STRING(1..5)   := "PCMXX";
ROUTE_TABLE_FILE         : STRING(1..56)  :=
     "APPL_DISK1:[VNIJEB]ROUTE_TABLE_FOR_EQUIPMENT_TYPE_XX.DAT";
--       12345678901234567890123456789012345678901234567890123456
--             1         2         3         4         5
ARGUMENT_STRING          : STRING(1..255);      -- Argument passed to MPIP
ARGUMENT_STRING_LENGTH   : INTEGER := 255;
DAY_STAMP                : STRING(1..2)   := (others => ' ');

TRACING_LOGICAL          : STRING(1..255) := (others => ' ');
TRACING_LOGICAL_LEN      : INTEGER := 0;
TRACE_ON                 : BOOLEAN := FALSE;

STATUS                   : VCU_STATUS_TYPE;    -- for VCU status.

MSG_SIZE                 : constant INTEGER
                                   := 4000;    -- Maximum size of message.
MAIL_BOX_SIZE            : constant INTEGER
                                   := 4000; -- 10000Maximum mail box size.
MESSAGE_HEADER_SIZE      : constant INTEGER
                                   := 14;      -- Size of message header.

pragma VOLATILE (DAY_STAMP);
pragma VOLATILE (APPL_PID);
```

```
    pragma VOLATILE (NO_OF_PORTS);
    pragma VOLATILE (PORT_DATA);
    pragma VOLATILE (GET_MBOX_STATUS);
--
-- exceptions
--
    ABORT_MPIP              : exception;

    package ROUTE_TABLE_IO is new INDEXED_IO(ROUTE_TABLE_RECORD);
    use ROUTE_TABLE_IO;

    ROUTE_TABLE_FILE_TYPE   : ROUTE_TABLE_IO.FILE_TYPE;

--
-- separates procedures
--
    procedure PUT_LOG(S: STRING ) is
    begin
        TEXT_IO.PUT_LINE(SYSTEM_ANSII_TIME & " " & S );
    end;
    pragma INLINE(PUT_LOG);

    procedure PUT_LOGICAL_MESSAGE(LOG : STRING;
                                  STR : STRING) is

    STAT : CONDITION_HANDLING.COND_VALUE_TYPE;

    begin
        RTL.LIB_SET_LOGICAL(RETURN_STATUS       => STAT,
                            LOG_NAME            => LOG,
                            VALUE               => SYSTEM_ANSII_TIME & " " &
                                                   STR,
                            TABLE_DESC          => "LNM$SYSTEM_TABLE");
    exception
        when others => PUT_LOG(CURRENT_EXCEPTION.NAME);
    end;
    pragma INLINE(PUT_LOGICAL_MESSAGE);

    procedure PUT_TRACE (S: STRING ) is separate;
    procedure READ_ID_BY_KEY is new READ_BY_KEY ( SHORT_INTEGER );
    procedure READ_PORT_ID_BY_KEY is new READ_BY_KEY ( SHORT_INTEGER );
    procedure GET_PARAMETERS is separate;
    procedure CREATE_ROUTE_TABLE is separate;
    procedure CREATE_AGENT_MAILBOX is separate;
    procedure SUSPENDED_WAIT is separate;
```

```
    GET_PARAMETERS;

    DAY_STAMP := SYSTEM_ANSII_TIME(7..8);

    TOTAL_PORTS := INTEGER'VALUE(NO_OF_PORTS);

    PUT_LOG("%MPIP-I, Received command_line parameters");

    NEW_MBOX( NAME => MPIP_MAIL_BOX_NAME,
              SIZE => MAIL_BOX_SIZE,
              STATUS => STATUS);

    if STATUS = ERROR then
        PUT_LOG("%MPIP-I, Unable to create mail box, Name "
                & MPIP_MAIL_BOX_NAME );
        raise ABORT_MPIP;
    end if;

    MBOX_LOCATE ( NAME    => MPIP_MAIL_BOX_NAME,
                  MBOX_POINTER  => MPIP_MBOX_PTR,
                  STATUS => STATUS );

    MPIP_ID := MPIP_MBOX_PTR.CHANNEL;

    CREATE_ROUTE_TABLE;
    PUT_LOG("%MPIP-I, Route table file created/initialized");

    CREATE_AGENT_MAILBOX;

    SUSPENDED_WAIT;          -- The process is suspended waiting for messages.

exception

    when ABORT_MPIP =>
        if IS_OPEN(FILE    => ROUTE_TABLE_FILE_TYPE ) then
            CLOSE ( FILE => ROUTE_TABLE_FILE_TYPE );
        end if;
        PUT_LOG("%MPIP-F, MPIP initialization failed. ");

end MPIP;
```

```
-- ++
-- FACILITY:
--
--          VAXSECS II COMMUNICATION UTILITY
--          GET_PORT_DATA
--
-- ABSTRACT:
--
--          This subroutine read data from the port.  It handles all the
--          protocol needed to communicate with the GEIC.
--
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
--          1/8/91 Rewritten version 1.2.
--                 Modified byte by byte QIO with 50 byte chunk QIO to
--                 read the port data.
--                 Streamlined code to improve readability.
--         3/10/90 Modify to suit MPIP.
--         8/10/90 Modify so that we send data by channel type rather
--                 than searching through the data structure for the port
--                 name!
--
separate (MPIP.SUSPENDED_WAIT)


procedure GET_PORT_DATA(
             PORT_CHANNEL: STARLET.CHANNEL_TYPE;
             MSG         : in out VCU_MSG_TYPE;
             TIMOUT      : INTEGER := 2;
             STATUS      : in out VCU_STATUS_TYPE ) is

 ENQB                : character;
 BYTE_BUFF           : SYSTEM.UNSIGNED_BYTE := 0;
 BYTE_ARRAY_BUFF     : MSG_ARR_TYPE(1..INTRA_BLOCK_CNT):=(others => 0);
 WORD_BIT_ARR        : SYSTEM.BIT_ARRAY_16;
 STAT                : CONDITION_HANDLING.COND_VALUE_TYPE;
 DSTAT               : VCU_STATUS_TYPE := NONE;
 IOSB                : STARLET.IOSB_TYPE;
 LOOP_COUNT          : INTEGER := 0; -- number of loops to do
 REMAINDER           : INTEGER := 0; -- remainder no of bytes to receive.
```

```
NO_DATA_RECEIVED    : exception;  -- failure to receive ENQ.
BAD_MESSAGE         : exception;  -- failure to receive message data.
BAD_MESSAGE_LENGTH  : exception;  -- failure to receive length bytes.

begin
-- PUT_TRACE("starting GET_PORT_DATA " & NAME);
   STATUS := NONE;
   MSG.MSG_LEN := 0;
--
--  Wait for ENQ
--
    loop
       TASKING_SERVICES.TASK_QIOW
                  (STATUS      => STAT,
                   CHAN        => PORT_CHANNEL,
                   FUNC        =>
                   SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                        integer(STARLET.IO_M_NOECHO)+
                                        integer(STARLET.IO_M_NOFILTR)+
                                        integer(STARLET.IO_M_TIMED)),
              IOSB     => IOSB,
              P1       => SYSTEM.TO_UNSIGNED_LONGWORD(ENQB'ADDRESS),
              P2       => SYSTEM.UNSIGNED_LONGWORD(1),
              P3       => SYSTEM.UNSIGNED_LONGWORD(TIMOUT),
              P4       => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

       if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
--
-- Timeout for ENQ.  We exited with status = NO_DATA.
--
          STATUS := NO_DATA;
          raise NO_DATA_RECEIVED;

       elsif not CONDITION_HANDLING.SUCCESS(STAT) then
--
-- Some other error condition in calling QIO.  We exited with status = ERROR
--
          PUT_LOG("MPIP: GET_PORT_DATA: Receive ENQ failed, "&
                  "Error status is " &
                  CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
          STATUS := ERROR;
          raise NO_DATA_RECEIVED;

       elsif CONDITION_HANDLING.SUCCESS(STAT) then
--
```

```
            TASKING_SERVICES.TASK_QIOW
                    ( STATUS  => STAT,
                      CHAN    => PORT_CHANNEL,
                      FUNC    => STARLET.IO_WRITELBLK,
                      IOSB    => IOSB,
                      P1      => SYSTEM.TO_UNSIGNED_LONGWORD(EOT'ADDRESS),
                      P2      => SYSTEM.UNSIGNED_LONGWORD(1));
               exit;
            else
               PUT_LOG
                 ("MPIP GET_PORT_DATA: Receive Other than ENQ - discarded");
            end if;
         end if;
      end loop;

--
-- We are ready to get the rest of the data.  First we obtain the high
-- byte of the message length.
--
      TASKING_SERVICES.TASK_QIOW
          (STATUS => STAT,
          CHAN   => PORT_CHANNEL,
          FUNC   =>
             SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                  integer(STARLET.IO_M_NOECHO)+
                                  integer(STARLET.IO_M_NOFILTR)+
                                  integer(STARLET.IO_M_TIMED)),
          IOSB   => IOSB,
          P1     => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_BUFF'ADDRESS),
          P2     => SYSTEM.UNSIGNED_LONGWORD(1),
           -- intercharacter timeout = 3s
          P3     => SYSTEM.UNSIGNED_LONGWORD(3),
          P4     => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

      if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
         STATUS := TIMEOUT;
         raise BAD_MESSAGE_LENGTH;

      elsif not CONDITION_HANDLING.SUCCESS(STAT) then
         PUT_LOG("MPIP GET_PORT_DATA: Recieve High byte of length failed, "&
                   "Error status is " &
                   CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
         STATUS := ERROR;
         raise BAD_MESSAGE_LENGTH;
```

```
--
-- and now to get the low byte.
--
    TASKING_SERVICES.TASK_QIOW
        (STATUS => STAT,
         CHAN   => PORT_CHANNEL,
         FUNC   =>
            SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                 integer(STARLET.IO_M_NOECHO)+
                                 integer(STARLET.IO_M_NOFILTR)+
                                 integer(STARLET.IO_M_TIMED)),
         IOSB   => IOSB,
         P1     => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_BUFF'ADDRESS),
         P2     => SYSTEM.UNSIGNED_LONGWORD(1),
         -- intercharacter timeout = 3s
         P3     => SYSTEM.UNSIGNED_LONGWORD(3),
         P4     => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

    if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
       STATUS := TIMEOUT;
       raise BAD_MESSAGE_LENGTH;

    elsif not CONDITION_HANDLING.SUCCESS(STAT) then
       PUT_LOG("MPIP GET_PORT_DATA: Recieve Low byte of length failed, "&
                 "Error status is " &
                  CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
       STATUS := ERROR;
       raise BAD_MESSAGE_LENGTH;

    else
--
-- We have obtained the appropriate byte length.  Now to convert it into
-- an integer.
--
       WORD_BIT_ARR(0..7) := SYSTEM.TO_BIT_ARRAY_8(BYTE_BUFF);
       MSG.MSG_LEN := INTEGER(SYSTEM.TO_UNSIGNED_WORD(WORD_BIT_ARR));

--       PUT_TRACE("VCU GET_PORT_DATA: MSG Recv len = " &
--                                       integer'image(MSG.MSG_LEN));
    end if;

--
-- Now to get the message proper.  In the code below, the data is obtained
-- in byte array of size INTRA_BLOCK_CNT.
```

```
   CURR_BYTE_POSITION := 1;

   for I in 1..LOOP_COUNT loop
      TASKING_SERVICES.TASK_QIOW
         (STATUS   => STAT,
          CHAN     => PORT_CHANNEL,
          FUNC     =>
              SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                   integer(STARLET.IO_M_NOECHO)+
                                   integer(STARLET.IO_M_NOFILTR)+
                                   integer(STARLET.IO_M_TIMED)),
          IOSB     => IOSB,
          P1       => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_ARRAY_BUFF'ADDRESS),
          P2       => SYSTEM.UNSIGNED_LONGWORD(INTRA_BLOCK_CNT),
       -- intercharacter timeout = 5s
          P3       => SYSTEM.UNSIGNED_LONGWORD(5),
          P4       => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

       if INTEGER(IOSB.STATUS) = STARLET.SS_TIMEOUT then
          PUT_LOG("MPIP GET_PORT_DATA: Timeout at block" &
                   INTEGER'IMAGE(I) & " Msg Len:" &
                   INTEGER'IMAGE(MSG.MSG_LEN) & " ***TIMEOUT***");
          STATUS := TIMEOUT;
          raise BAD_MESSAGE;

       elsif not CONDITION_HANDLING.SUCCESS(STAT) then
          PUT_LOG("MPIP GET_PORT_DATA: Recieve MESSAGE failed, "&
                  "Error status is " &
                  CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
          STATUS := ERROR;
          raise BAD_MESSAGE;

       else
--
-- OK, now we transfer the data to the message buffer, update the byte
-- position, and send out an ACK.
-- Next, continue to loop to get the next 50 byte array.
--
          NEW_BYTE_POSITION := CURR_BYTE_POSITION + INTRA_BLOCK_CNT;
          MSG.MSG(CURR_BYTE_POSITION..
                    NEW_BYTE_POSITION - 1 ) := BYTE_ARRAY_BUFF;
          CURR_BYTE_POSITION := NEW_BYTE_POSITION ;

          TASKING_SERVICES.TASK_QIOW
                (STATUS => STAT,
```

```
                P1      => SYSTEM.TO_UNSIGNED_LONGWORD(ACK'ADDRESS),
                P2      => SYSTEM.UNSIGNED_LONGWORD(1));

      end if;
   end loop;

--
-- We now check whether we have anything else left to read.
-- If not we quited the procedure.
--
   if REMAINDER = 0 then
      STATUS := DONE; -- ie finished receiving all the stuff.
   else
      TASKING_SERVICES.TASK_QIOW
         (STATUS   => STAT,
          CHAN     => PORT_CHANNEL,
          FUNC     =>
             SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                  integer(STARLET.IO_M_NOECHO)+
                                  integer(STARLET.IO_M_NOFILTR)+
                                  integer(STARLET.IO_M_TIMED)),
          IOSB     => IOSB,
          P1       => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_ARRAY_BUFF'ADDRESS),
          P2       => SYSTEM.UNSIGNED_LONGWORD(REMAINDER),
       -- intercharacter timeout = 5s
          P3       => SYSTEM.UNSIGNED_LONGWORD(5),
          P4       => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

      if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
         STATUS := TIMEOUT;
         raise BAD_MESSAGE;

      elsif not CONDITION_HANDLING.SUCCESS(STAT) then
         PUT_LOG("MPIP GET_PORT_DATA: Recieve MESSAGE failed, "&
                 "Error status is " &
                 CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
         STATUS := ERROR;
         raise BAD_MESSAGE;

      else
--
-- OK, now we transfer the rest of the data to the message buffer,
-- and send an ACK.
--
         MSG.MSG(CURR_BYTE_POSITION..
```

```
            TASKING_SERVICES.TASK_QIOW
                  (STATUS => STAT,
                   CHAN   => PORT_CHANNEL,
                   FUNC   => STARLET.IO_WRITELBLK,
                   IOSB   => IOSB,
                   P1     => SYSTEM.TO_UNSIGNED_LONGWORD(ACK'ADDRESS),
                   P2     => SYSTEM.UNSIGNED_LONGWORD(1));
            STATUS := DONE;
         end if;
      end if;
--
-- if logical is defined then MSG is dump out
--
      STARLET.TRNLNM
          ( STATUS        => STAT,
            TABNAM        => "LNM$PROCESS_TABLE",
            LOGNAM        => "VCU$LOG_RECEIVE_MSG");

      if integer(STAT) = integer(STARLET.SS_NORMAL) then

          PUT_LOG("Receive Message from PORT " &
                UNSIGNED_WORD'IMAGE(PORT_CHANNEL) & " DUMP");

          DUMP_MESSAGE(   MSG     => MSG,
                          STATUS  => DSTAT);
      end if;
   exception
      when PORT_NOT_FOUND | NO_DATA_RECEIVED | BAD_MESSAGE_LENGTH => null;
      when BAD_MESSAGE =>
   --
   -- We will send out a nak to indicate our displeasure with the whole
   -- transaction!
   --
          TASKING_SERVICES.TASK_QIOW
                  (STATUS => STAT,
                   CHAN   => PORT_CHANNEL,
                   FUNC   => STARLET.IO_WRITELBLK,
                   IOSB   => IOSB,
                   P1     => SYSTEM.TO_UNSIGNED_LONGWORD(NAK'ADDRESS),
                   P2     => SYSTEM.UNSIGNED_LONGWORD(1));
      when others =>
          PUT_LOG("MPIP GET_PORT_DATA:" & CURRENT_EXCEPTION.NAME);
          raise;
   end GET_PORT_DATA;
   pragma SUPPRESS_ALL;
```

```
-- ++
-- FACILITY:
--
--          MULTI_PORT_INTERFACE_PROCESS( MPIP )
--          ATTACH_EQ_ID_TO_LIST
--
-- ABSTRACT:
--
--          This procedure updates the eq id linked list attached
--          to the PIP.
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
-- --
separate (MPIP.SUSPENDED_WAIT.PROCESS_PORT_MESSAGE)

procedure ALL_EQUIPMENT_ONLINE_MESSAGE is

    READ_BUFFER         : ROUTE_TABLE_RECORD;
    WRITE_BUFFER        : ROUTE_TABLE_RECORD;
    CURR_EQ_ID          : SYSTEM.UNSIGNED_BYTE := 0;
    DATA_TYPE           : AMF_DATA_TYPE;
    DATA_TYPE_LEN       : INTEGER := 0;
    STATUS              : VCU_STATUS_TYPE := NONE;
    LIST_LENGTH         : INTEGER := 0;

begin
    INCOMING_MESSAGE.POS := 14;

    GET_ITEM_FORMAT( MSG => INCOMING_MESSAGE,
                     TYP => DATA_TYPE,
                     LEN => LIST_LENGTH,
                     STATUS => STATUS );

    for J in 1.. LIST_LENGTH loop
        GET_ITEM_FORMAT ( MSG => INCOMING_MESSAGE,
                          TYP => DATA_TYPE,
                          LEN => DATA_TYPE_LEN,
                          STATUS => STATUS );

        GET_UNSIGNED_BYTE ( MSG => INCOMING_MESSAGE,
```

```
        WRITE_BUFFER.PORT_ID := SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID);
        WRITE_BUFFER.EQ_ID   := SHORT_INTEGER(CURR_EQ_ID);
        WRITE_BUFFER.INDEX   := SHORT_INTEGER(INDEX);

        begin
            WRITE  ( FILE   => ROUTE_TABLE_FILE_TYPE,
                     ITEM   => WRITE_BUFFER );

            PUT_LOG("%MPIP-I-ONLINE, Eq_Type" & UNSIGNED_BYTE'IMAGE(EQ_TYPE)
                & " id" & UNSIGNED_BYTE'IMAGE(CURR_EQ_ID) &
                " online.");

        exception
            when ROUTE_TABLE_IO.KEY_ERROR =>
              READ_ID_BY_KEY ( FILE => ROUTE_TABLE_FILE_TYPE,
                               ITEM => READ_BUFFER,
                               KEY  => WRITE_BUFFER.EQ_ID,
                               KEY_NUMBER => 0,
                               RELATION   => EQUAL );

              ROUTE_TABLE_IO.UNLOCK(ROUTE_TABLE_FILE_TYPE);

              if READ_BUFFER.PORT_ID /=
                    SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID) then
--
-- duplicated equipment found!
--
                  DUPLICATED_ID_DETECTED( CURR_EQ_ID );
              end if;
        end;
    end loop;
end ALL_EQUIPMENT_ONLINE_MESSAGE;
```

```
-- ++
-- FACILITY:
--
--          PROCESS CONTROL MANAGER (PCM)
--          - PROCESS_APPLICATION_MESSAGE__CLOSE_SESSION;
--
-- ABSTRACT:
--
--          This message tells the PCM that the application is closing session.
--          We get the ID, then kills off the mailbox that is associated with
--          it.
--
--
-- AUTHOR:
--
--          SYSTEMS SOFTWARE, AUTOMATION DEPARTMENT, NSEB.
--
-- MODIFICATION HISTORY:
--
--          ORIGINAL VERSION, 1 - AUGUST -1991
--          Version 0.1
--
separate (MPIP.SUSPENDED_WAIT.PROCESS_APPL_MESSAGE)

procedure APPL_CLOSE_SESSION is

APPL_MBOX_PTR       :  MBOX_ACC_TYPE;
APPL_MBOX_NAME      :  STRING(1..8);

begin

    MBOX_LOCATE ( CHANNEL      => APPL_ID,
                  MBOX_POINTER => APPL_MBOX_PTR,
                  STATUS       => STATUS );

    if MBOX_POINTER = NULL then
        return;
    end if;

    APPL_MBOX_NAME  := APPL_MBOX_PTR.NAME.all;

    REMOVE_MBOX ( NAME   =>  APPL_MBOX_NAME,
                  STATUS =>  STATUS );

    for K in 1..NO_OF_APPL loop
```

```
        end if;
    end loop;

    PUT_LOG ("%MPIP-CLOSES-I, Application "
              & APPL_MBOX_NAME & " has closed session");

exception
    when others =>
        PUT_LOG("%MPIP-W-EXC, Exception is " & CURRENT_EXCEPTION.NAME );
end APPL_CLOSE_SESSION;
```

```
    end if;

    if APPL_PID(K) = 0 then
        APPL_PID(K) := PID_HEXSTR_TO_DEC(APPL_PID_STRING);
        ASSIGN := TRUE;
        exit;
    end if;
end loop;

if not ASSIGN then

    if NO_OF_APPL = 20 then
        PUT_LOG
           ("%MPIP-W-TOOMANYAPPL, Too many application logon, try later");
        OPEN_FAILED := TRUE;
    else
        NO_OF_APPL := NO_OF_APPL + 1;
        APPL_PID(NO_OF_APPL) := PID_HEXSTR_TO_DEC(APPL_PID_STRING);
    end if;
end if;

NEW_MBOX ( NAME   => APPL_PID_STRING,
           STATUS => STATUS );

MBOX_LOCATE ( NAME   => APPL_PID_STRING,
              MBOX_POINTER => CHANNEL_PTR,
              STATUS       => STATUS );

APPL_ID := CHANNEL_PTR.CHANNEL;

if OPEN_FAILED then
    APPL_ID := 0;
end if;

EQ_TYPE := UNSIGNED_BYTE'VALUE(MACHINE_TYPE);

BUILD_HEADER ( MSG        => OUT_MESSAGE,
               APPL_ID    => APPL_ID,
               PIP_ID     => 0,
               XPC_ID     => 0,
               EQ_TYPE    => EQ_TYPE,
               EQ_ID      => 0,
               STREAM     => 140,
               FUNCTN     => 22,
               REPLY_FLAG => NO_REPLY,
```

```
                SYSTEM_BYTE_3  => 0,
                SYSTEM_BYTE_4  => 0 );


   SEND_MESSAGE ( MSG       => OUT_MESSAGE,
                  CHANNEL   => APPL_ID,
                  STATUS    => STATUS );

   if (STATUS = NOT_FOUND ) then
       PUT_LOG("%MPIP-W, Unable to open application session");
       REMOVE_MBOX  ( NAME      => APPL_PID_STRING,
                      STATUS    => STATUS );
   else
       if not OPEN_FAILED then
          PUT_LOG("%MPIP-I, Application open session, PID is " &
                   APPL_PID_STRING );
       else
          REMOVE_MBOX ( NAME    => APPL_PID_STRING,
                        STATUS  => STATUS );
       end if;
    end if;

exception
    when others =>
        PUT_LOG("%MPIP-OPNSES, Exception is " & CURRENT_EXCEPTION.NAME );
end APPL_OPEN_SESSION;
```

```
-- ++
-- FACILITY:
--
--          PROCESS CONTROL MANAGER
--          CREATE_AGENT_MAILBOX
--
-- ABSTRACT:
--
--          This stub is used to provide a way for user to customize
--          PCM, so that appropriate agents mailbox may be created when
--          needed.
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
-- --
separate (MPIP)
procedure CREATE_AGENT_MAILBOX is
--
-- Currently this file does nothing.
--
--    [declarative_part]
begin

    NEW_MBOX( NAME => "KNS1482_MBOX",
              SIZE => MAIL_BOX_SIZE,
              STATUS => STATUS);

    if STATUS = ERROR then
        PUT_LOG("%PCM-I, Unable to create mail box, Name "
                & "KNS1482_MBOX" );
        raise ABORT_MPIP;
    end if;
    if STATUS = DONE then
       PUT_LOG ("%PCM-I, To create mail box, name kns1482_mbox");
    end if;

end CREATE_AGENT_MAILBOX;
```

```
--
-- MODIFICATION HISTORY:
--
-- --
separate (MPIP)
procedure CREATE_AGENT_MAILBOX is
--
-- Currently this file does nothing.
--
--    [declarative_part]
begin

    NEW_MBOX( NAME => "KNS1482_MBOX",
              SIZE => MAIL_BOX_SIZE,
              STATUS => STATUS);

    if STATUS = ERROR then
        PUT_LOG("%PCM-I, Unable to create mail box, Name "
                & "KNS1482_MBOX" );
        raise ABORT_MPIP;
    end if;
    if STATUS = DONE then
       PUT_LOG ("%PCM-I, To create mail box, name kns1482_mbox");
    end if;

end CREATE_AGENT_MAILBOX;
```

```
-- ++
-- FACILITY:
--
--          MULTI PORT INTERFACE PROCESS
--          CREATE_ROUTE_TABLE
--
-- ABSTRACT:
--
--          Subroutine to create the route table file.
--
--          -
-- AUTHOR:
--
--          SYSTEMS SOFTWARE, AUTOMATION DEPARTMENT, NSEB.
--
-- MODIFICATION HISTORY:
--
--          ORIGINAL VERSION, 11 - JULY -1991
--          Version 0.1
--
separate (MPIP)

procedure CREATE_ROUTE_TABLE is

    ROUTE_TABLE_FORM_STRING : constant STRING :=
        "FILE;"                                          &
            "ORGANIZATION  INDEXED;"                     &
            "PROTECTION (S:RWED,O:RWED,G:RWE,W:RWE);" &
            "GLOBAL_BUFFER_COUNT 6;"                     &
        "RECORD;"                                        &
            "BLOCK_SPAN YES;"                            &
            "FORMAT FIX;"                                &
        "ACCESS;"                                        &
            "GET TRUE;"                                  &
            "PUT TRUE;"                                  &
            "DELETE TRUE;"                               &
            "UPDATE TRUE;"                               &
        "SHARING;"                                       &
            "GET TRUE;"                                  &
            "PUT TRUE;"                                  &
            "DELETE TRUE;"                               &
            "UPDATE TRUE;"                               &
            "PROHIBIT FALSE;"                            &
        "CONNECT;"                                       &
            "WAIT_FOR_RECORD YES;"                       &
```

```
            "BEST_TRY_CONTIGUOUS YES;"          &
            "BUCKET_SIZE 3;"                    &
            "EXTENSION 3;"                      &
        "AREA 1;"                               &
            "ALLOCATION 6;"                     &
            "BEST_TRY_CONTIGUOUS YES;"          &
            "BUCKET_SIZE 3;"                    &
            "EXTENSION 3;"                      &
        "AREA 2;"                               &
            "ALLOCATION 36;"                    &
            "BEST_TRY_CONTIGUOUS YES;"          &
            "BUCKET_SIZE 3;"                    &
            "EXTENSION 12;"                     &
        "KEY 0;"                                &
          "DATA_AREA 0;"                        &
          "DATA_FILL 100;"                      &
          "DATA_KEY_COMPRESSION NO;"            &
          "DATA_RECORD_COMPRESSION NO;"         &
          "INDEX_AREA 1;"                       &
          "INDEX_COMPRESSION NO;"               &
          "INDEX_FILL 100;"                     &
          "LEVEL1_INDEX_AREA 1;"                &
          "LENGTH 2;"                           &
          "POSITION 0;"                         &
          "DUPLICATES       NO;"                &
          "TYPE             INT2;"              &
        "KEY 1;"                                &
          "DATA_AREA 2;"                        &
          "DATA_FILL 100;"                      &
          "DATA_KEY_COMPRESSION NO;"            &
          "INDEX_AREA 2;"                       &
          "INDEX_COMPRESSION NO;"               &
          "INDEX_FILL 100;"                     &
          "LEVEL1_INDEX_AREA 2;"                &
          "LENGTH 2;"                           &
          "POSITION 2;"                         &
          "CHANGES          YES;"               &
          "DUPLICATES       YES;"               &
          "TYPE             INT2";
begin
    ROUTE_TABLE_FILE(51..52) := MACHINE_TYPE;

    ROUTE_TABLE_IO.CREATE( FILE => ROUTE_TABLE_FILE_TYPE,
                           MODE => ROUTE_TABLE_IO.INOUT_FILE,
                           NAME => ROUTE_TABLE_FILE,
```

```
    when ROUTE_TABLE_IO.USE_ERROR =>
        PUT_LOG ("%MPIP-CRETBL-F, Failed to create route table file ");
        raise ABORT_MPIP;

end CREATE_ROUTE_TABLE;
```

```
-- ++
-- FACILITY:
--
--          PROCESS CONTROL MANAGER (PCM)
--          - PROCESS_PIP_MESSAGE__DO_S151_005
--
-- ABSTRACT:
--
--          This routine is called when a particular machine/xpc
--          becomes offline.
--          The information is deleted from the route table file.
--
-- AUTHOR:
--
--           SYSTEMS SOFTWARE, AUTOMATION DEPARTMENT, NSEB.
--
-- MODIFICATION HISTORY:
--
--          ORIGINAL VERSION, 24 - JULY -1990.
--          Version 0.1
--          13/02/91 - Modify module to delete records also when eq id = 0,
--                     which means the XPC has become offline.
--
separate (MPIP.SUSPENDED_WAIT.PROCESS_PORT_MESSAGE)


procedure EQUIPMENT_OFFLINE_MESSAGE is

--
-- If we receive this message, we will delete the necessary info from
-- out route table data base.
--
    READ_BUFFER : ROUTE_TABLE_RECORD;

begin
--
-- We now need to check whether the eq id is 0.  If so, then the XPC has gone
-- offline, and we will need to rid of all the equipment with that XPC.
--
    if SHORT_INTEGER(EQ_ID) /= 0 then
       begin

          READ_BUFFER.EQ_ID  := SHORT_INTEGER(EQ_ID);

          READ_ID_BY_KEY ( FILE        => ROUTE_TABLE_FILE_TYPE,
```

```
                        RELATION    => ROUTE_TABLE_IO.EQUAL );

        if READ_BUFFER.PORT_ID = SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID) then
            DELETE_ELEMENT( FILE => ROUTE_TABLE_FILE_TYPE );

            PUT_LOG ("%MPIP-I-OFFLINE, Eq_Type" &
            UNSIGNED_BYTE'IMAGE(EQ_TYPE) &
            " Id" & UNSIGNED_BYTE'IMAGE(EQ_ID) & " offline ");

        else
            PUT_LOG ("%MPIP-I-ALOFF, Equipment already offline");
            UNLOCK( ROUTE_TABLE_FILE_TYPE );

        end if;



      exception
--
-- Our offline machine has already been deleted ...
--
         when ROUTE_TABLE_IO.EXISTENCE_ERROR =>
            PUT_LOG ("%MPIP-I-ALOFF, Eq_Type" & UNSIGNED_BYTE'IMAGE(EQ_TYPE)
             & " Id" & UNSIGNED_BYTE'IMAGE(EQ_ID)
             & " already offline ");

         when OTHERS   => raise;
      end;

   else     -- we try to offline all equipment attached to the XPC.
        REMOVE_ALL_ID(INDEX);
   end if;
end EQUIPMENT_OFFLINE_MESSAGE;
```

```
-- ++
-- FACILITY:
--
--          MULTI_PORT_INTERFACE_PROCESS (MPIP)
--          EQUIPMENT_ONLINE_MESSAGE
--
-- ABSTRACT:
--
--          This module process an equipment online message, by updating th
--          internal data structure as well as the route table file.
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
-- --
separate(MPIP.SUSPENDED_WAIT.PROCESS_PORT_MESSAGE)

procedure EQUIPMENT_ONLINE_MESSAGE is

WRITE_BUFFER    : ROUTE_TABLE_RECORD;
READ_BUFFER     : ROUTE_TABLE_RECORD;

begin

    WRITE_BUFFER.EQ_ID   := SHORT_INTEGER(EQ_ID);
    WRITE_BUFFER.PORT_ID := SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID);
    WRITE_BUFFER.INDEX   := SHORT_INTEGER(INDEX);

    ROUTE_TABLE_IO.WRITE
                ( FILE          => ROUTE_TABLE_FILE_TYPE,
                  ITEM          => WRITE_BUFFER );

    PUT_LOG("%MPIP-I-ONLINE, Eq_Type" &
            UNSIGNED_BYTE'IMAGE(EQ_TYPE) &
            "'id'," & UNSIGNED_BYTE'IMAGE(EQ_ID) & " online. ");

    SEND_MESSAGE_TO_APPL;

exception

    when ROUTE_TABLE_IO.KEY_ERROR =>
```

```
                KEY        => WRITE_BUFFER.EQ_ID,
                KEY_NUMBER => 0,
                RELATION   => ROUTE_TABLE_IO.EQUAL );

        ROUTE_TABLE_IO.UNLOCK (ROUTE_TABLE_FILE_TYPE);

        if READ_BUFFER.PORT_ID /=
                    SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID) then

            PUT_LOG("%MPIP-W-DUPID, Found duplicated eq_id, " & "XPC is "
                      & SHORT_INTEGER'IMAGE(READ_BUFFER.PORT_ID) );

            DUPLICATED_ID_DETECTED( EQ_ID );

            raise EXIT_PROCESS_MESSAGE;

        else
            PUT_LOG("%MPIP-I-ALRON, EQ " & UNSIGNED_BYTE'IMAGE(EQ_ID) &
                    " already online ");
            raise EXIT_PROCESS_MESSAGE;

        end if;

end EQUIPMENT_ONLINE_MESSAGE;
```

```
-- ++
-- FACILITY:
--
--          MULTI_PORT_INTERFACE_PROCESS (MPIP)
--          EQUIPMENT_ONLINE_MESSAGE
--
-- ABSTRACT:
--
--          This module process an equipment online message, by updating th
--          internal data structure as well as the route table file.
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
-- --
separate(MPIP.SUSPENDED_WAIT.PROCESS_PORT_MESSAGE)

procedure EQUIPMENT_ONLINE_MESSAGE is

WRITE_BUFFER    : ROUTE_TABLE_RECORD;
READ_BUFFER     : ROUTE_TABLE_RECORD;

begin

    WRITE_BUFFER.EQ_ID   := SHORT_INTEGER(EQ_ID);
    WRITE_BUFFER.PORT_ID := SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID);
    WRITE_BUFFER.INDEX   := SHORT_INTEGER(INDEX);

    ROUTE_TABLE_IO.WRITE
                ( FILE          => ROUTE_TABLE_FILE_TYPE,
                  ITEM          => WRITE_BUFFER );

    PUT_LOG("%MPIP-I-ONLINE, Eq_Type" &
            UNSIGNED_BYTE'IMAGE(EQ_TYPE) &
            " id" & UNSIGNED_BYTE'IMAGE(EQ_ID) & " online. ");

    SEND_MESSAGE_TO_APPL;

exception

    when ROUTE_TABLE_IO.KEY_ERROR =>
```

```
                KEY        => WRITE_BUFFER.EQ_ID,
                KEY_NUMBER => 0,
                RELATION   => ROUTE_TABLE_IO.EQUAL );

        ROUTE_TABLE_IO.UNLOCK (ROUTE_TABLE_FILE_TYPE);

        if READ_BUFFER.PORT_ID /=
                    SHORT_INTEGER(PORT_DATA(INDEX).XPC_ID) then

            PUT_LOG("%MPIP-W-DUPID, Found duplicated eq_id, " & "XPC is "
                        & SHORT_INTEGER'IMAGE(READ_BUFFER.PORT_ID) );

            DUPLICATED_ID_DETECTED( EQ_ID );

            raise EXIT_PROCESS_MESSAGE;

        else
            PUT_LOG("%MPIP-I-ALRON, EQ " & UNSIGNED_BYTE'IMAGE(EQ_ID) &
                        " already online ");
            raise EXIT_PROCESS_MESSAGE;

        end if;

end EQUIPMENT_ONLINE_MESSAGE;
```

```
-- ++
-- FACILITY:
--
--          MULTI PORT INTERFACE PROCESS
--          GET_PARAMETERS
--
-- ABSTRACT:
--
--          This subroutine does initialization by using FORIN to
--          to get arguments for PCM.  The arguments are MACHINE_TYPE,
--          NO_OF_PORTS and actual port_name.
--
-- AUTHOR:
--
--          Systems group, Automation Department, NSEB
--
-- MODIFICATION HISTORY:
--
--           Original Version.
--
separate (MPIP)

procedure GET_PARAMETERS is

begin
    FORIN.GET_COMMAND( STR     => ARGUMENT_STRING,
                       STR_LEN => ARGUMENT_STRING_LENGTH);

    FORIN.GET_PARAMETER(INSTR => ARGUMENT_STRING(1..ARGUMENT_STRING_LENGTH),
                        PARM  => MACHINE_TYPE,
                        PARM_LEN => MACHINE_TYPE_LENGTH);

    MPIP_MAIL_BOX_NAME(4..5) := MACHINE_TYPE;

    FORIN.GET_PARAMETER(INSTR => ARGUMENT_STRING(1..ARGUMENT_STRING_LENGTH),
                        PARM => TRACING_LOGICAL,
                        PARM_LEN => TRACING_LOGICAL_LEN);

    TRACING_LOGICAL(TRACING_LOGICAL_LEN + 1.. TRACING_LOGICAL_LEN + 18 ) :=
                                        "_MPIP$TRACE_ENABLE";

    TRACING_LOGICAL_LEN := TRACING_LOGICAL_LEN + 18;

    FORIN.GET_PARAMETER(INSTR => ARGUMENT_STRING(1..ARGUMENT_STRING_LENGTH),
                        PARM => NO_OF_PORTS,
```

```
-- ++
-- FACILITY:
--
--          VAXSECS II COMMUNICATION UTILITY
--          GET_PORT_DATA
--
-- ABSTRACT:
--
--          This subroutine read data from the port.  It handles all the
--          protocol needed to communicate with the GEIC.
--
--
-- AUTHOR:
--
--          System Software, NSEB CIM.
--
-- MODIFICATION HISTORY:
--
--          1/8/90 Rewritten version 1.2.
--                 Modified byte by byte QIO with 50 byte chunk QIO to
--                 read the port data.
--                 Streamlined code to improve readability.
--         3/10/90 Modify to suit MPIP.
--         8/10/90 Modify so that we send data by channel type rather
--                 than searching through the data structure for the port
--                 name!
--
separate (MPIP.SUSPENDED_WAIT)


procedure GET_PORT_DATA(
            PORT_CHANNEL: STARLET.CHANNEL_TYPE;
            MSG         : in out VCU_MSG_TYPE;
            TIMOUT      : INTEGER := 2;
            STATUS      : in out VCU_STATUS_TYPE ) is

ENQB                : character;
BYTE_BUFF           : SYSTEM.UNSIGNED_BYTE := 0;
BYTE_ARRAY_BUFF     : MSG_ARR_TYPE(1..INTRA_BLOCK_CNT):=(others => 0);
WORD_BIT_ARR        : SYSTEM.BIT_ARRAY_16;
STAT                : CONDITION_HANDLING.COND_VALUE_TYPE;
DSTAT               : VCU_STATUS_TYPE := NONE;
IOSB                : STARLET.IOSB_TYPE;
LOOP_COUNT          : INTEGER := 0; -- number of loops to do
REMAINDER           : INTEGER := 0; -- remainder no of bytes to receive.
```

```
NO_DATA_RECEIVED    : exception;  -- failure to receive ENQ.
BAD_MESSAGE         : exception;  -- failure to receive message data.
BAD_MESSAGE_LENGTH  : exception;  -- failure to receive length bytes.

begin
-- PUT_TRACE("starting GET_PORT_DATA " & NAME);
   STATUS := NONE;
   MSG.MSG_LEN := 0;
--
--  Wait for ENQ
--
    loop
       TASKING_SERVICES.TASK_QIOW
                  (STATUS      => STAT,
                   CHAN        => PORT_CHANNEL,
                   FUNC        =>
                   SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                        integer(STARLET.IO_M_NOECHO)+
                                        integer(STARLET.IO_M_NOFILTR)+
                                        integer(STARLET.IO_M_TIMED)),
                IOSB     => IOSB,
                P1       => SYSTEM.TO_UNSIGNED_LONGWORD(ENQB'ADDRESS),
                P2       => SYSTEM.UNSIGNED_LONGWORD(1),
                P3       => SYSTEM.UNSIGNED_LONGWORD(TIMOUT),
                P4       => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

       if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
--
-- Timeout for ENQ.  We exited with status = NO_DATA.
--
          STATUS := NO_DATA;
          raise NO_DATA_RECEIVED;

       elsif not CONDITION_HANDLING.SUCCESS(STAT) then
--
-- Some other error condition in calling QIO.  We exited with status = ERR
--
          PUT_LOG("MPIP: GET_PORT_DATA: Receive ENQ failed, "&
                  "Error status is " &
                  CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
          STATUS := ERROR;
          raise NO_DATA_RECEIVED;

       elsif CONDITION_HANDLING.SUCCESS(STAT) then
--
```

```
            TASKING_SERVICES.TASK_QIOW
                    ( STATUS  => STAT,
                      CHAN    => PORT_CHANNEL,
                      FUNC    => STARLET.IO_WRITELBLK,
                      IOSB    => IOSB,
                      P1      => SYSTEM.TO_UNSIGNED_LONGWORD(EOT'ADDRESS
                      P2      => SYSTEM.UNSIGNED_LONGWORD(1));
               exit;
            else
               PUT_LOG
                 ("MPIP GET_PORT_DATA: Receive Other than ENQ - discarded");
            end if;
         end if;
      end loop;

   --
   -- We are ready to get the rest of the data.  First we obtain the high
   -- byte of the message length.
   --
      TASKING_SERVICES.TASK_QIOW
          (STATUS => STAT,
           CHAN   => PORT_CHANNEL,
           FUNC   =>
              SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                   integer(STARLET.IO_M_NOECHO)+
                                   integer(STARLET.IO_M_NOFILTR)+
                                   integer(STARLET.IO_M_TIMED)),
           IOSB   => IOSB,
           P1     => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_BUFF'ADDRESS),
           P2     => SYSTEM.UNSIGNED_LONGWORD(1),
            -- intercharacter timeout = 3s
           P3     => SYSTEM.UNSIGNED_LONGWORD(3),
           P4     => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

      if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
         STATUS := TIMEOUT;
         raise BAD_MESSAGE_LENGTH;

      elsif not CONDITION_HANDLING.SUCCESS(STAT) then
         PUT_LOG("MPIP GET_PORT_DATA: Recieve High byte of length failed, "&
                   "Error status is " &
                   CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
         STATUS := ERROR;
         raise BAD_MESSAGE_LENGTH;
```

```
--
-- and now to get the low byte.
--
    TASKING_SERVICES.TASK_QIOW
        (STATUS => STAT,
         CHAN   => PORT_CHANNEL,
         FUNC   =>
            SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                 integer(STARLET.IO_M_NOECHO)+
                                 integer(STARLET.IO_M_NOFILTR)+
                                 integer(STARLET.IO_M_TIMED)),
         IOSB   => IOSB,
         P1     => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_BUFF'ADDRESS),
         P2     => SYSTEM.UNSIGNED_LONGWORD(1),
         -- intercharacter timeout = 3s
         P3     => SYSTEM.UNSIGNED_LONGWORD(3),
         P4     => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

    if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
       STATUS := TIMEOUT;
       raise BAD_MESSAGE_LENGTH;

    elsif not CONDITION_HANDLING.SUCCESS(STAT) then
       PUT_LOG("MPIP GET_PORT_DATA: Recieve Low byte of length failed, "&
                  "Error status is " &
                  CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
       STATUS := ERROR;
       raise BAD_MESSAGE_LENGTH;

    else
--
-- We have obtained the appropriate byte length.  Now to convert it into
-- an integer.
--
       WORD_BIT_ARR(0..7) := SYSTEM.TO_BIT_ARRAY_8(BYTE_BUFF);
       MSG.MSG_LEN := INTEGER(SYSTEM.TO_UNSIGNED_WORD(WORD_BIT_ARR));

--       PUT_TRACE("VCU GET_PORT_DATA: MSG Recv len = " &
--                                      integer'image(MSG.MSG_LEN));
    end if;

--
-- Now to get the message proper.  In the code below, the data is obtained
-- in byte array of size INTRA_BLOCK_CNT.
```

```
CURR_BYTE_POSITION ;= 1;

for I in 1..LOOP_COUNT loop
   TASKING_SERVICES.TASK_QIOW
      (STATUS   => STAT,
       CHAN     => PORT_CHANNEL,
       FUNC     =>
           SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                integer(STARLET.IO_M_NOECHO)+
                                integer(STARLET.IO_M_NOFILTR)+
                                integer(STARLET.IO_M_TIMED)),
       IOSB     => IOSB,
       P1       => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_ARRAY_BUFF'ADDRESS
       P2       => SYSTEM.UNSIGNED_LONGWORD(INTRA_BLOCK_CNT),
     -- intercharacter timeout = 5s
       P3       => SYSTEM.UNSIGNED_LONGWORD(5),
       P4       => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

    if INTEGER(IOSB.STATUS) = STARLET.SS_TIMEOUT then
       PUT_LOG("MPIP GET_PORT_DATA: Timeout at block" &
                INTEGER'IMAGE(I) & " Msg Len:" &
                INTEGER'IMAGE(MSG.MSG_LEN) & " ***TIMEOUT***");
       STATUS := TIMEOUT;
       raise BAD_MESSAGE;

    elsif not CONDITION_HANDLING.SUCCESS(STAT) then
       PUT_LOG("MPIP GET_PORT_DATA: Recieve MESSAGE failed, "&
               "Error status is " &
               CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
       STATUS := ERROR;
       raise BAD_MESSAGE;

    else
--
-- OK, now we transfer the data to the message buffer, update the byte
-- position, and send out an ACK.
-- Next, continue to loop to get the next 50 byte array.
--
       NEW_BYTE_POSITION := CURR_BYTE_POSITION + INTRA_BLOCK_CNT;
       MSG.MSG(CURR_BYTE_POSITION..
                NEW_BYTE_POSITION - 1 ) := BYTE_ARRAY_BUFF;
       CURR_BYTE_POSITION := NEW_BYTE_POSITION ;

       TASKING_SERVICES.TASK_QIOW
            (STATUS => STAT,
```

```
                P1      => SYSTEM.TO_UNSIGNED_LONGWORD(ACK'ADDRESS),
                P2      => SYSTEM.UNSIGNED_LONGWORD(1));

      end if;
   end loop;

--
-- We now check whether we have anything else left to read.
-- If not we quited the procedure.
--
   if REMAINDER = 0 then
      STATUS := DONE; -- ie finished receiving all the stuff.
   else
      TASKING_SERVICES.TASK_QIOW
         (STATUS    => STAT,
          CHAN      => PORT_CHANNEL,
          FUNC      =>
             SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                  integer(STARLET.IO_M_NOECHO)+
                                  integer(STARLET.IO_M_NOFILTR)+
                                  integer(STARLET.IO_M_TIMED)),
          IOSB      => IOSB,
          P1        => SYSTEM.TO_UNSIGNED_LONGWORD(BYTE_ARRAY_BUFF'ADDRESS
          P2        => SYSTEM.UNSIGNED_LONGWORD(REMAINDER),
       -- intercharacter timeout = 5s
          P3        => SYSTEM.UNSIGNED_LONGWORD(5),
          P4        => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

      if integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
         STATUS := TIMEOUT;
         raise BAD_MESSAGE;

      elsif not CONDITION_HANDLING.SUCCESS(STAT) then
         PUT_LOG("MPIP GET_PORT_DATA: Recieve MESSAGE failed, "&
                 "Error status is " &
                 CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
         STATUS := ERROR;
         raise BAD_MESSAGE;

      else
--
-- OK, now we transfer the rest of the data to the message buffer,
-- and send an ACK.
--
         MSG.MSG(CURR_BYTE_POSITION..
```

```
-- ++
-- FACILITY:
--
--         Vax Secs II - VSII
--
-- ABSTRACT:
--
--         The equipment id/XPC id received is unrecognized.  We will
--         send a S159 F001 message back to the port for actions.  Note
--         that this is done by going through the PIP_MBX.
--
-- AUTHOR:
--
--         System Software, NSEB.
--
-- MODIFICATION HISTORY:
--
--        17/1/90 - Changed the header such that the machine id is 0
--                  This is because the message is destined for XPC.
--
--
separate (MPIP.SUSPENDED_WAIT)

procedure ILLEGAL_ID_DETECTED( INCOMING_MESSAGE : VCU_MSG_TYPE;
                               INDEX            : INTEGER ) is

   OUTGOING_MSG : VCU_MSG_TYPE(50);
   STATUS       : VCU_STATUS_TYPE;

begin

  REMOVE_ALL_ID(INDEX);

  BUILD_HEADER ( MSG        => OUTGOING_MSG,
                 APPL_ID    => 0,
                 PIP_ID     => MPIP_ID,
                 XPC_ID     => PORT_DATA(INDEX).XPC_ID,
                 EQ_TYPE    => UNSIGNED_BYTE'VALUE(MACHINE_TYPE),
                 EQ_ID      => 0,
                 STREAM     => 159,
                 FUNCTN     => 1,
                 REPLY_FLAG => NO_REPLY,
                 DIRECTION_FLAG => VAX_TO_EQ,
                 SYSTEM_BYTE_1 => 0,
                 SYSTEM_BYTE_2 => 0,
```

```
    loop

        FORIN.GET_PARAMETER(INSTR => ARGUMENT_STRING(1..ARGUMENT_STRING_LENGTH),
                            PARM  => PORT_DATA(J).PORT_NAME,
                            PARM_LEN => PORT_DATA(J).PORT_NAME_LEN );

    end loop;

end GET_PARAMETERS;
```

```
   PUT_ITEM_FORMAT ( MSG => OUTGOING_MSG,
                     TYP => BINARY,
                     LEN => 14,
                     STATUS => STATUS );

   OUTGOING_MSG.MSG(17..30) := INCOMING_MESSAGE.MSG(1..14);
   OUTGOING_MSG.MSG_LEN := 30;
   OUTGOING_MSG.POS := 30;

   PUT_MBOX_DATA ( NAME => MPIP_MAIL_BOX_NAME,
                   MSG  => OUTGOING_MSG,
                   STATUS => STATUS );

end ILLEGAL_ID_DETECTED;
```

```
-- ++
-- FACILITY:
--
--          PORT_INTERFACE_PROCESS
--          MBOX_IN_TASK
--
-- ABSTRACT:
--
--          MBOX_IN - process messages from PCM.
--          This task is nearly independent from the other tasks.  It
--          reads the mailbox, and process the messages accordingly.
--          Most of the time, it will be forwarding the messages to the
--          port.  This task also sends message to the port on behalf of
--          of PORT_IN.
--
-- AUTHOR:
--
--          System Software, NSEB Automation
--
-- MODIFICATION HISTORY:
--
--          05/02/90  Original Version.
--
separate (MPIP.SUSPENDED_WAIT)

task body MBOX_IN_TASK is

INCOMING_MESSAGE        : VCU_MSG_TYPE(MSG_SIZE);
MSG_OUT_STATUS          : VCU_STATUS_TYPE := NONE;
STATUS                  : VCU_STATUS_TYPE := NONE;

begin
   loop
      GET_MBOX_DATA ( NAME     => MPIP_MAIL_BOX_NAME,
                      MSG      => INCOMING_MESSAGE,
                      WAIT     => TRUE,
                      STATUS   => GET_MBOX_STATUS  );

      if GET_MBOX_STATUS = DONE then

         PROCESS_APPL_MESSAGE ( INCOMING_MESSAGE => INCOMING_MESSAGE );

      end if;
   end loop;
end;
```

```
-- ++
-- FACILITY:
--
--          VAXSECS II COMMUNICATIONS UTILITY (VCU)
--          PUT_PORT_DATA
--
-- ABSTRACT:
--
--          This procedure puts out message in the message buffer, and
--          handles all port protocol with XPC.  In case of any failure,
--          the routine will do a resend up to a default of 3 times before
--          it exits.  Returns a status of ERROR,DONE,TIMEOUT or
--          PORT_NOT_FOUND.
--
--          DONE           - normal completion of PUT_PORT_DATA.
--          PORT_NOT_FOUND - port name has not been assigned a channel.
--          TIMEOUT        - timeout in receiving EOT or ACK.
--          ERROR          - undefined error in sending data to port.
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
--          31/7/90 Modified code to put out 50 bytes at a time to speed
--                  up QIO, and reduces CPU usage compared to previous code
-- --
separate (MPIP.SUSPENDED_WAIT)

procedure PUT_PORT_DATA
        ( PORT_CHANNEL  : STARLET.CHANNEL_TYPE;
          MSG           : in out VCU_MSG_TYPE;
          RESET_MSG     : BOOLEAN := TRUE;
          RETRY         : INTEGER := 3;
          TIMOUT        : INTEGER := 8;
          STATUS        : in out VCU_STATUS_TYPE) is

 STAT                   : CONDITION_HANDLING.COND_VALUE_TYPE;
 DSTAT                  : VCU_STATUS_TYPE;
 IOSB                   : STARLET.IOSB_TYPE;
 PORT_CUR_PTR           : PORT_ACC_TYPE := PORT_HEAD_PTR;
 EOTB                   : CHARACTER;
 ACKB                   : CHARACTER;
 MSGW                   : SYSTEM.UNSIGNED_WORD := 0;
```

```
LOOP_COUNT              : INTEGER := 0;
REMAINDER               : INTEGER := 0;

PORT_NOT_FOUND          : exception;
BAD_SEND                : exception;
RETRY_SEND              : exception;

begin
   STATUS := NONE;
--
-- Send an ENQ to see if it RECEIVER  will accept our ENQ.
--
   MAIN: loop
      begin
         TASKING_SERVICES.TASK_QIOW
                ( STATUS    => STAT,
                  CHAN      => PORT_CHANNEL,
                  FUNC      => STARLET.IO_WRITELBLK,
                  IOSB      => IOSB,
                  P1        => SYSTEM.TO_UNSIGNED_LONGWORD(ENQ'ADDRESS),
                  P2        => SYSTEM.UNSIGNED_LONGWORD(1));
--
--  Must receive an EOT, if not then ERROR
--                      if ENQ then CONTENTION
--                      if TIMEOUT then retry
--
         EOTB := ' ';
         TASKING_SERVICES.TASK_QIOW
                ( STATUS    => STAT,
                  CHAN      => PORT_CHANNEL,
                  FUNC      =>
                  SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                       integer(STARLET.IO_M_NOECHO)+
                                       integer(STARLET.IO_M_NOFILTR)+
                                       integer(STARLET.IO_M_TIMED)),
                  IOSB      => IOSB,
                  P1        => SYSTEM.TO_UNSIGNED_LONGWORD(EOTB'ADDRESS),
                  P2        => SYSTEM.UNSIGNED_LONGWORD(1),
                  P3        => SYSTEM.UNSIGNED_LONGWORD(TIMOUT),
                  P4        => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS)

         if EOTB = EOT then
--
-- We received an EOT, therfore we can go ahead and send data
-- We first send the length of message (a word), by breaking up
```

```
            MSGW_BIT_ARRAY := SYSTEM.TO_BIT_ARRAY_16(MSGW);
--
-- We first send the Hi byte
--
            TASKING_SERVICES.TASK_QIOW
               ( STATUS   => STAT,
                 CHAN     => PORT_CHANNEL,
                 FUNC     => STARLET.IO_WRITELBLK,
                 IOSB     => IOSB,
                 P1       =>
                 SYSTEM.TO_UNSIGNED_LONGWORD(MSGW_BIT_ARRAY(8..15)'ADDRESS
                 P2       => SYSTEM.UNSIGNED_LONGWORD(1));

            if not CONDITION_HANDLING.SUCCESS(STAT) then
              PUT_LOG
                 ("MPIP PUT_PORT_DATA: Unsuccessful send HI length byte,"
                 "Error status is " &
                 CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
              STATUS := ERROR;
              raise RETRY_SEND;
            end if;
--
-- Now to send the LO byte.
--
            TASKING_SERVICES.TASK_QIOW
              ( STATUS   => STAT,
                CHAN     => PORT_CHANNEL,
                FUNC     => STARLET.IO_WRITELBLK,
                IOSB     => IOSB,
                P1       =>
                SYSTEM.TO_UNSIGNED_LONGWORD(MSGW_BIT_ARRAY(0..7)'ADDRESS
                P2       => SYSTEM.UNSIGNED_LONGWORD(1));

            if not CONDITION_HANDLING.SUCCESS(STAT) then
              PUT_LOG
                ("MPIP PUT_PORT_DATA: Unsuccessful send LO length byte,"
                 "Error status is " &
                 CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
              STATUS := ERROR;
              raise RETRY_SEND;
            end if;
--
-- Now to send the data in 50 byte chunks.  There is no clear reason why
-- we should do this except it seems a nice round number!
-- First we find out how many of those 50 bytes chunks we have, and also
```

```
-- other reasons, we do retries up to a specified time, and then we exited
--
            LOOP_COUNT := MSG.MSG_LEN / INTRA_BLOCK_CNT;
            REMAINDER  := MSG.MSG_LEN mod INTRA_BLOCK_CNT;
            CURR_BYTE_POSITION := 1;

            for I in 1..LOOP_COUNT loop
               TASKING_SERVICES.TASK_QIOW
                  (STATUS   => STAT,
                   CHAN     => PORT_CHANNEL,
                   FUNC     => STARLET.IO_WRITELBLK,
                   IOSB     => IOSB,
                   P1       => SYSTEM.TO_UNSIGNED_LONGWORD(
                                  MSG.MSG(CURR_BYTE_POSITION)'ADDRESS),
                   P2       => SYSTEM.UNSIGNED_LONGWORD(INTRA_BLOCK_CNT));

               CURR_BYTE_POSITION := CURR_BYTE_POSITION + INTRA_BLOCK_CNT;

               if not CONDITION_HANDLING.SUCCESS(STAT) then
--
-- if we failed to send, we try again.
--
                  PUT_LOG("MPIP: PUT_PORT_DATA, Unsuccessful send, "&
                  "Error status is " &
                  CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
                  STATUS := ERROR;
                  raise RETRY_SEND;
               end if;  -- unsuccessful send
            end loop;   -- of sending data
--
-- Now we check for any remainder, if any.  If so, we will finish sending
-- the remainder.
--
            if REMAINDER /= 0 then
               TASKING_SERVICES.TASK_QIOW
                  (STATUS   => STAT,
                   CHAN     => PORT_CHANNEL,
                   FUNC     => STARLET.IO_WRITELBLK,
                   IOSB     => IOSB,
                   P1       => SYSTEM.TO_UNSIGNED_LONGWORD(
                                  MSG.MSG(CURR_BYTE_POSITION)'ADDRESS),
                   P2       => SYSTEM.UNSIGNED_LONGWORD(REMAINDER));

               if not CONDITION_HANDLING.SUCCESS(STAT) then
--
```

```
                "Error status is " &
                CONDITION_HANDLING.COND_VALUE_TYPE'IMAGE(STAT) );
                STATUS := ERROR;
                raise RETRY_SEND;
             end if;  -- unsuccessful send
          end if;     -- remainder.
--
-- We have send out the message successfully.  Now to wait for an ACK.
--
          ACKB := ' ';

          TASKING_SERVICES.TASK_QIOW
             (STATUS  => STAT,
              CHAN    => PORT_CHANNEL,
              FUNC    =>
             SYSTEM.UNSIGNED_WORD(integer(STARLET.IO_READLBLK)+
                                  integer(STARLET.IO_M_NOECHO)+
                                  integer(STARLET.IO_M_NOFILTR)+
                                  integer(STARLET.IO_M_TIMED)),
              IOSB    => IOSB,
              P1      => SYSTEM.TO_UNSIGNED_LONGWORD(ACKB'ADDRESS),
              P2      => SYSTEM.UNSIGNED_LONGWORD(1),
              P3      => SYSTEM.UNSIGNED_LONGWORD(TIMOUT),
              P4      => SYSTEM.TO_UNSIGNED_LONGWORD(P4MASK'ADDRESS));

          if ACKB = ACK then
--
-- We have received an ACK to our message, which means that
-- everything is OK, the GEIC has successfully obtained all the data.
-- if logical is defined then MSG is dump out
--
             STARLET.TRNLNM(  STATUS  => STAT,
                              TABNAM  => "LNM$PROCESS_TABLE",
                              LOGNAM  => "VCU$LOG_SEND_MSG");

             if integer(STAT) = integer(STARLET.SS_NORMAL) then
                PUT_LOG("MPIP: PUT_PORT_DATA, Send Message to PORT "
                         & UNSIGNED_WORD'IMAGE(PORT_CHANNEL) & " DU

                DUMP_MESSAGE(MSG    => MSG,
                             STATUS => DSTAT);
             end if;

             STATUS := DONE;
             if RESET_MSG then
```

```
                exit;

            elsif integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
--
-- We timed out to receive an ACK.  Hence, we will retry sending.
--
                STATUS := TIMEOUT;
                raise RETRY_SEND;
            else
--
-- We did not receive an ACK but something else!  Hence, we will
-- retry sending.
--
                STATUS := ERROR;
                raise RETRY_SEND;
            end if;

        elsif EOTB = ENQ then
--
-- We have received an ENQ, which means XPC is going to send.  Therefore
-- we give way to him to send.
--
            STATUS := CONTENTION;
            exit;

        elsif integer(IOSB.STATUS) = STARLET.SS_TIMEOUT then
--
-- We timeout waiting for an EOT, hence we will retry.
--
            STATUS := TIMEOUT;
            raise RETRY_SEND;


        else
--
--  Error, expected data received
--
            STATUS := ERROR;
            raise RETRY_SEND;
        end if;

      exception
         when RETRY_SEND =>
            if RETRY_CNT = RETRY then
               raise BAD_SEND;
```

```
            when others => raise;

         end;
      end loop MAIN;

exception
   when PORT_NOT_FOUND => null;
   when BAD_SEND => null;
   when others =>
      PUT_LOG("MPIP: PUT_PORT_DATA, " & CURRENT_EXCEPTION.NAME);
      raise;
end PUT_PORT_DATA;
pragma SUPPRESS_ALL;
```

```
-- ++
-- FACILITY:
--
--          PROCESS CONTROL MANAGER (PCM)
--          SEND_MESSAGE
--
-- ABSTRACT:
--
--          This subroutine sends data to mailbox by channel.
--
-- AUTHOR:
--
--          System Software, NSEB
--
-- MODIFICATION HISTORY:
--
-- --
separate (MPIP.SUSPENDED_WAIT)

procedure SEND_MESSAGE( MSG     : in out VCU_MSG_TYPE;
                        CHANNEL : in STARLET.CHANNEL_TYPE;
                        STATUS  : in out VCU_STATUS_TYPE ) is

    MBOXP   : MBOX_ACC_TYPE := NULL;

begin

    STATUS := ERROR;

    MBOX_LOCATE (       CHANNEL     => CHANNEL,
                        MBOX_POINTER => MBOXP,
                        STATUS      => STATUS );

    if STATUS = NOT_FOUND then
        return;
    else
        PUT_MBOX_DATA(  NAME    => MBOXP.NAME.ALL,
                        MSG     => MSG,
                        STATUS  => STATUS );
    end if;

exception
    when others =>
        raise;
end SEND_MESSAGE;
```

บทสรุป และ วิจารณ์

จากผลการทำวิจัยทางคณะผู้จัดทำมีความเห็นว่า ระบบ PROTOCOL ที่ใช้ใน PROJECT นี้ สามารถนำไปพัฒนาใช้ได้กับระบบ CONTROL FREE RUNING อื่นได้ โดยไม่ต้องขึ้นอยู่กับด้านของ HARDWARE ซึ่งอาจจะใช้ระบบ INFRAFEAD , RADIO FGEQUENCY , หรือระบบสื่อสารที่ใช้สัญญานอย่างเช่น RS232, IEE488 เป็นต้นข้อดีของ SEC RPOTOCOL ที่ใช้ใน PROJECT นี้คือ สามารถส่งข้อมูลได้ทั้งหมด เช่น TEXT, INTERQER ขนาด 1,2,4,8 BYTE FLOTING POINT ขนาด 1,2,4,8, BYTE และ UNSINGNED ขนาดต่าง ๆ ในตัว PROTOCOL ออกแบบรองรับการสื่อสารจาก HOST ไป ถึงEQIUQMENTได้ทั้งหมด 2^15 EQUIPMENTซึ่งเป็นจำนวนมากมายที่สามารถรับคำสั่งจาก HOST COMPUTER 1 ตัว และทุกตัวของ EQUIPMENT ก็สามารถรายงานตัวกับ HOST ได้ ตลอดเวลา โดยระบบ QUEUE

ในด้านของข้อดีนี้สามารถดัดแปลงเช่น ต้องการควบคุมแขนกล จำนวน 10 ตัว โดยทุกตัวติดต่อผ่าน RADIO FREQUENCY ภายใน PROTOCOL SECSจะสามารถควบคุมการสั่งงานต่าง ๆ ได้จาก HOST ตัวอย่างเช่น HOST ต้องการสั่งให้แขนกลตัวที่ 8 จับชิ้นงานที่กำหนดไว้ใน PROGRAM จาก HOST การทำงานจะเริ่มโดย HOST ทำการ BUILD PROTOCOL ประกอบไปด้วย EQUIPMENT NUMBER&FLAG ในการที่จะบอกกับ EQUIPMENT ว่าต้องการ REPLY ข้อมูลกลับมาจาก EQUIPMENT หรือไม่ข้อมูลหมายเลขของ FUNCTION ซึ่งจุดนี้จะเป็นการกำหนดกันไว้ก่อนระหว่าง EQUIPMENT กับ HOST ตามด้วยความยาวของข้อมูลชนิดและเนื้อหาของข้อมูลตามสุดท้ายด้วย CHECK SUM ซึ่งในครั้งแรกทุก EQUIPMENT จะรับข้อมูลนี้ได้ทั้งหมดทุกตัวแต่ต่อมาถ้า NEMBER ของ EQUIPMENT นั้นไม่ตรงกับ EQUIPMENT ID ของตัวเองก็จะไม่สนใจข้อมูลนั้น ในแต่ละครั้งของการติดต่อภายใต้ SEC PROTOCOL สามารถส่งข้อมูลได้สูงสุด 152 KBYTE ซึ่งพอเพียงสำหรับ ระบบควบคุมแบบ FREERUNNING และเป็นการก้าวเข้าสู่ระะ CIM อย่างแท้จริง และยังเป็นการไม่จำเป็นต้องระบุชนิดของ COMPUTER เพื่อนำการควบคุมเนื่องจาก การควบคุมอยู่ที่มาตรฐาน RPOTOCOL

หนังสืออ้างอิง

1. DIGITAL EQUIPMENT CORPORATE MASSACHUSETTS, Vax Ada language Reference Manual, Digital U.S.A, May 1989.
2. DIGITAL EQUIPMENT CORPORATE MASSACHUSETTS, Vax Ada Developing Program, Digital U.S.A, May 1989.
3. DIGITAL EQUIPMENT CORPORATE MASSACHUSETTS, Vax Ada Run-Time Reference, Digital U.S.A, May 1989.
4. DIGITAL EQUIPMENT CORPORATE MASSACHUSETTS, Vax Ada VMS Network Management V7, Digital U.S.A, Apr 1988.
5. DIGITAL EQUIPMENT CORPORATE MASSACHUSETTS, Vax Ada VMS Run-Time Referance V7, Digital U.S.A, Apr 1988.
6. National Semiconductor Corporate, Series 32000 data Book, National SC, Feb 1988.
7. Norman H.Cohen, Ada as a second language, Mcgraw_hill Book company, Mar 1987.

++++++++++++++++++++++++++++