HITACHI
S10α SERIES

SOFTWARE MANUAL
GENERAL DESCRIPTION & MACROS

COMPACT PMS V5

2α SERIES

Applicable to :
HITACHI-S10/2α
HITACHI-S10/2α E
HITACHI-S10/2α H
HITACHI-S10/2α Hf

HITACHI

NOTE

All information in this manual is based on the latest product information available at the time of printing. Hitachi has reviewed the accuracy of this manual, but assumes no responsibility for any omissions or errors which may appear. The design of the product is under constant review and, while every effort is made to keep this manual up to date, the right is reserved to change specifications and equipment at any time without prior notice.

PROHIBITION

These products should not be used for medical, power supply, nuclear, water supply, drainage plants, traffic control, military, space, nor disaster prevention equipment.

Diversion and/or resale of these products without this manual is prohibited.

Reproduction of the contents of this manual in whole or in part, without written permission of Hitachi, is prohibited.

TRADEMARKS

HITACHI－S10／2α, S10／4α and PSEα are registered trademarks of Hitachi, Ltd.

FIRST EDITION, December, 1990, SP - 63 - 220 (out of print)
SECOND EDITION, December, 1995, SAE - 3 - 201 (A)

LIMITED WARRANTY

Hitachi, Ltd., warrants its products to be manufactured in accordance with published specifications and free from defects in materials and/or workmanship.

Hitachi, Ltd., warrants its products against defects in parts and workmanship for one full year from date of purchase.

HITACHI, LTD., MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED EXCEPT AS PROVIDED HEREIN, INCLUDING WITHOUT LIMITATION THEREOF, WARRANTIES AS TO MARKETABILITY FOR A PARTICULAR PURPOSE OF USE, OR AGAINST INFRINGEMENT OF ANY PATENT. IN NO EVENT SHALL HITACHI BE LIABLE FOR ANY DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY NATURE, OR COSTS, CHARGES, LOSSES OR EXPENSES RESULTING FROM ANY DEFECTIVE PRODUCT OR THE USE OF ANY PRODUCT.

SOFTWARE UP-TO DATE POLICY

Hitachi, Ltd., constantly reviews its software so as to incorporate the latest technology. Hitachi reserves the right to make changes to any software to improve reliability, function, or design. Hitachi cannot be held responsible for any errors in its software.

SAFETY AWARENESS SUMMARY

The following are general safety precautions which must be observed in the application, operation, and maintenance of this equipment. Failure to comply with these precautions or the other caution statements in the manuals violates safety standards of design, manufacture, and intended use. Hitachi assumes no liability for the user's failure to comply with these requirements. This summary, and the caution statements in the manuals, represent warnings of certain dangers of which we are aware. You, as the end user of the equipment, must follow these warnings and all other applicable precautions, including codes and laws, to achieve safe application and operation of this equipment.

Safety Disconnects

As outlined in the manuals, you must provide means to disable the control and power circuits to guard against unexpected or sudden motion or energization of equipment during operation and maintenance. NEVER WORK ON WIRING WHICH IS ENERGIZED.

Care in Programming and Precautions Against Equipment Failure

The user must follow procedures as indicated in the manuals and as dictated by sound engineering judgment. Mistakes in programming may result in sudden or unexpected motion or energization. To protect against programming errors or equipment failure, you must provide physical guards and cages to prevent physical contact with equipment, and back-up safety equipment independent of the programmable controller; the latter includes overspeed protection, overtemperature protection, and electro-mechanical stop switches. NEVER DEPEND ON SOFTWARE OR CONTROLS TO PROTECT PERSONNEL WITHOUT PREPARING APPROPRIATE LOCKOUTS AND EQUIPMENT GUARDS.

Warning Devices

The user should provide audible and visual warning devices to warn persons to get clear of machines before they start. The user must properly program the programmable controller to operate these devices before the machine starts.

Environmental Requirements

This equipment is not suitable for use in an explosive atmosphere. If inputs or outputs are wired to devices in an explosive atmosphere, you must insert appropriate approved electrical barriers in the wiring conduit, install the equipment in explosion—proof cabinets and wire the installation according to the appropriate electrical code (ex. National Electric Code.) The other environmental requirements in the manuals must also be met, otherwise equipment failure could cause personal injury or property damage.

Do Not Service or Adjust Internal Parts

Personal injury may result from unauthorized servicing or adjusting parts inside the cabinets.

Prevent Spillage of Liquid onto the Equipment

Personal injury could result if any liquid is spilled or poured onto this equipment. The equipment is general purpose (NEMA Type 1 ventilated) and not waterproof.

Prevent Entry of Foreign Matter into the Equipment

Permitting metal chips and/or other foreign matter to enter the equipment could cause a short–circuit that could result in personal injury or property damage.

Keep the Plant Free of Vermin

Rodents, like rats and mice, may chew on cables and equipment. This could cause personal injury or property damage.

Do not Install the Equipment Near Strong Magnetic Fields

Operating the equipment near a strong magnetic field could cause malfunctions that could result in personal injury or property damage.

Protect From Shock and Vibration

Subjecting the equipment to shock or vibration could cause malfunctions that could result in personal injury or property damage.

Dangerous Voltages

Dangerous voltages are present whether the equipment is running or not. These voltages could be inside the programmable controller enclosure or in external control devices.

Danger of Manually Operating Limit Switches or Pushbuttons

Never operate a limit switch by hand. The resulting motion could cause personal injury. If you plan to operate a limit switch, be certain that you are clear of any other moving parts, then use a long wooden pole. Do not operate a pushbutton during checkout or at any other time unless you are sure what action the pushbutton causes, and are sure nobody is near any part that might move or be energized unexpectedly.

“RUN⁄STOP” SWITCH CAUTION

The “RUN⁄STOP” switch only stops execution of the ladder logic program or Hi–Flow program. Digital and analog outputs are left in the active state when execution stops, unless the optional rungs described in the CPU manual have been added. The “RUN⁄ STOP” switch does not affect the operation of C–language or FA–BASIC language programs. Outputs can still be produced in response to C–language or FA–BASIC programs, or by the action of programmers typing in commands in these languages, while the “RUN/STOP” switch is in the “STOP” position.

DO NOT DEPEND ON THE STOP SWITCH TO STOP MOVING PARTS OR TO PREVENT UNEXPECTED MOTION OR ENERGIZATION. USE HARDWIRED SAFETY STOPPING DEVICES, AS EXPLAINED IN THE CPU MANUAL. ALWAYS DISCONNECT AND LOCK OUT POWER AND CONTROL VOLTAGES BEFORE WORKING ON ELECTRICAL CIRCUITS OR PARTS THAT CAN MOVE.

General Specifications

Supply voltage		100-120 VAC, single-phase 50/60 Hz±4 Hz
Supply voltage range		85-132 VAC
Permissible duration of momentary power failure		10 ms or less (at rated input)
Temperature	Operational	32 to 131 °F (0 to 55 ℃)
	Storage	—4 to 158 °F (-20 to 70 ℃)
Humidity	Operational	30—90% RH
	Storage	10—90% RH
Vibration resistance (Max)		0.6 G (1000 rpm)
Impact resistance (Max)		10 G
Electrical noise tolerance		Noise Voltage 1,200 Vpp Noise duration 1 μsec Noise frequency 50 Hz
Voltage resistance		1,500 VAC, 1 min. between each external AC terminal and case
Insulation resistance		5 MΩ or more as measured with 500 VDC insulation resistance meter between each external AC terminal and case
Resistance to ground		Less than 100 ohms
Dust/gases		0.1 mg／m^3 or less; no corrosive gas permitted
Cooling method		Natural cooling

Programming Terminal PSEα Specifications

Supply voltage		100−120 VAC $^{+10}_{-15}$ % single-phase 50/60 Hz±4 Hz	
Power requirement	Continuous	130 VA	
	Surge	6,000 VA	
Temperature		Operational	Storage
		50 to 95 °F (10 to 35 ℃)	23 to 122 °F (−5 to +50 ℃)
Humidity		40−80% RH	10−98% RH
Vibration (Max)		0.5 G, 17 Hz vibration applied for 30 s	
Dust		0.1 mg/m^3 or less	
Dimensions	EL cover closed	400 W×110 H×350 D (mm)	
	EL cover open	400 W×230 H×350 D (mm)	
Weight		Approx. 4.5 kg (10 lb)	

PREFACE

Thank you for purchasing this HITACHI programmable controller (PC).

This manual describes the Compact PMS (CPMS) operating system that enhances the real-time and multitasking capabilities of the HITACHI S10/2α sequencer series.

CPMS gives the 2α series of controllers power comparable to that of full-scale control computer in many applications.

<Manual organization>

1. General Description

   This chapter describes the overall configuration of Compact PMS (CPMS).

2. Task Management

   This chapter describes task scheduling, task operations, and other functions required to construct a realtime system.

3. System Management

   This chapter describes built-in subroutines used for system starting up, error handling, and user-specific processing.

4. Macro Instruction Specifications

   This chapter describes the specifications of macro instructions as well as how to link them.

Appendixes

   These appendixes provide debugging and macroprogramming facilities listings and supplementary information.

This manual is applicable to the following system floppy disks:

Target tool	System floppy disk name	Version
PSEα	Compact PMS SYS	Version 5.0, Revision 0.0 or later
PS/2	CPMS Load System	Version 4.2, Revision 0.0 or later
	CPMSE Load System	Version 2.2, Revision 0.0 or later

The following table shows the relationships between CPMS versions and supported macro instructions.

Macro instruction	CPMS versions		
	V1	V4	V5
rleas	√	√	√
queue	√	√	√
abort	√	√	√
delay	√	√	√
timer	√	√	√ *1
ctime	√	√	√
chap	√	√	√
chmod	√	√	√
sfact	√	√	√
gfact	√	√	√
uspchk	√	√	√
stime		√	√
gtime		√	√
wake		√	√
cwake		√	√
rserv		√	√
free		√	√
mvmem		√	√

√ : Supported
*1 Additional features are available.

CONTENTS

1 GENERAL DESCRIPTION

1 GENERAL DESCRIPTION

1.1 CPMS

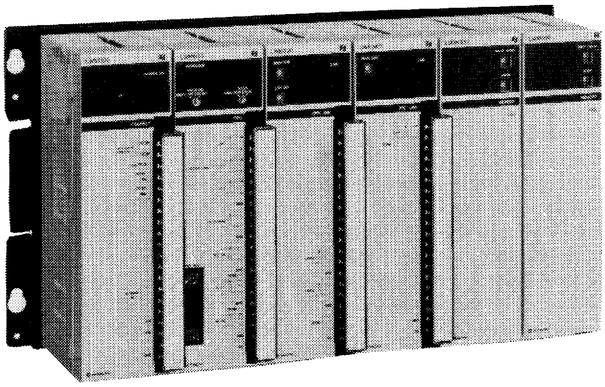
CPMS is a realtime OS.

CPMS has various improved functions for control systems that handles, in realtime, signals changing as the time elapses.

1.2 Hardware

Expansion memory is required in order to use CPMS. User-created programs and the debugger that debugs programs are stored in expansion memory. Select a memory size according to the size of programs.

If the HI FLOW system is planned to be used in the future, do not use the first 256K bytes for the user-created programs.



In the figure, CPU-to-CPU and PSE linkage option modules are mounted. (They are not required.)

■ Configuration

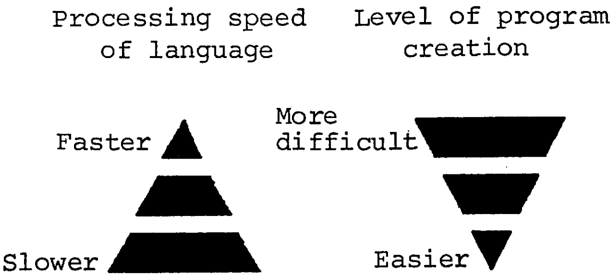
Expansion mount base	: HPC-1000
CPU power supply module	: LWV000
CPU module	: LWP000(2 α)
Expansion memory	: LWM

1.3 Software

1.3.1 Programs

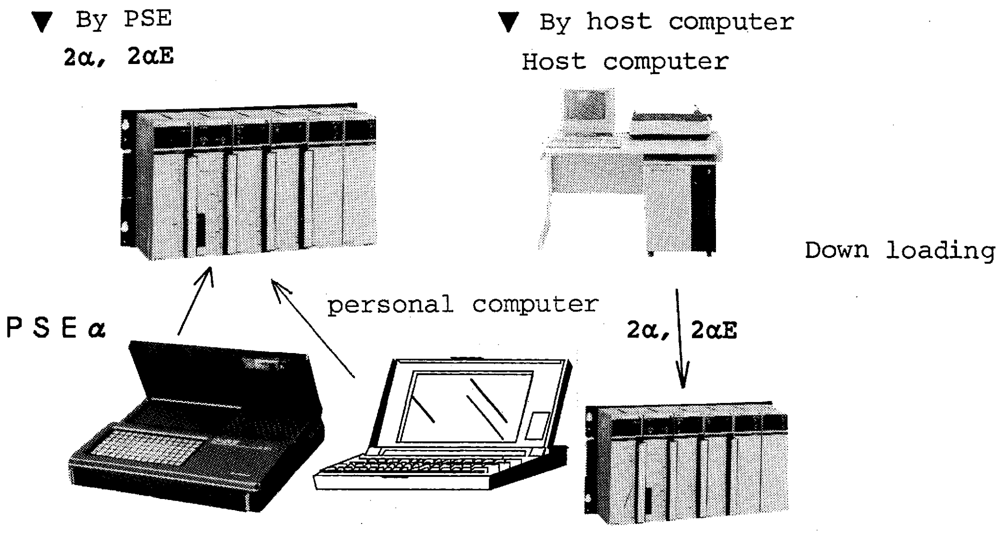
The three program languages listed below can be used to create programs that can be executed under CPMS. Select one according to the system processing time and how easy programs can be created.

- ■ 68000 assembler language
- ■ C language
- ■ FA-BASIC language



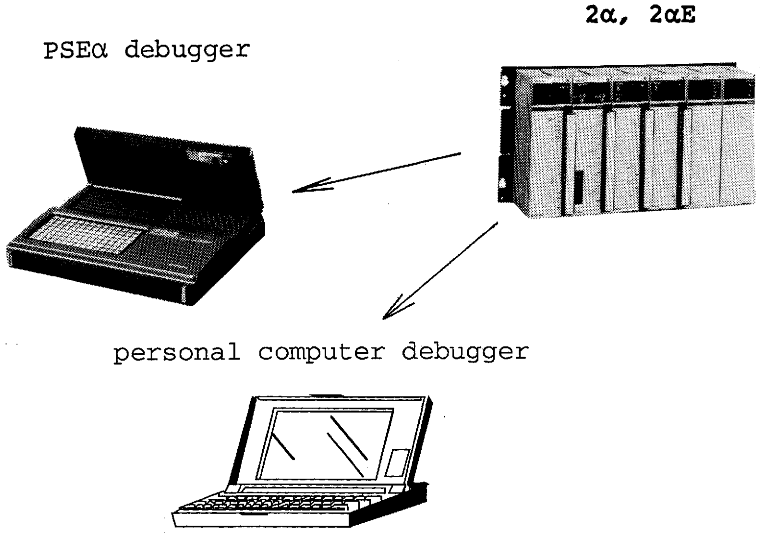
1.3.2 Loading programs

Programs are loaded by PSEα or a host computer such as V90 series using H-7338 host computer linkage.



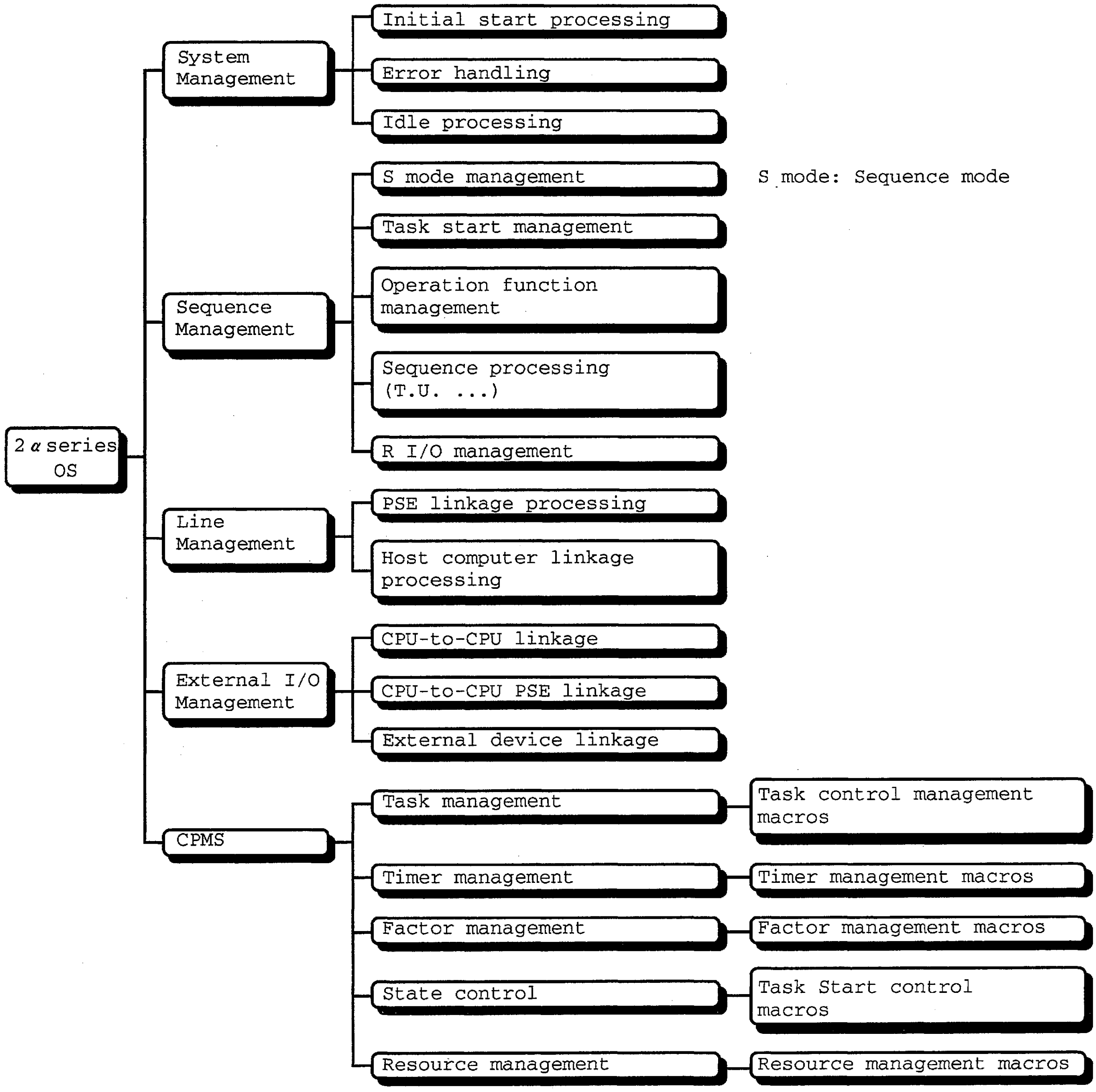
1.3.3 Checking programs

Programs created by the user can be checked by the PSE debugger.



1.4 Relationship between 2α OS and CPMS

CPMS is the realtime OS that operates under 2α series (2α, 2αE, 2αH, 2αHf) OS. The CPMS manages the operations of tasks created by the user.



S mode: Sequence mode

●Task: Minimum program unit used in the flow of control performed under CPMS.

■ Initial start processing

When the power supply is turned on, initial start processing initializes the system tables and hardware registers.

■ Error handling

If a CPU error occurs, error handling collects the information of the error and enables other operations.

■ Idle processing

If a CPU error or hardware error occurs, idle processing converts the information of the error to a code and indicates it on the CPU console LED. Idle processing also counts time.

■ S mode management

S mode management manages starting and restarting of S mode programs.

■ Task start management

Task start management schedules tasks started by the P coil (Started by a S mode program).

■ Operation function management

Operation function management executes operation functions started by operation instructions.

■ Sequence processing

Sequence processing controls the timer (T) and one shot (U) required for sequence control.

■ RI/O management

RI/O management receives the RI/O transfer termination, collects RI/O state data, and performs start operation.

■ PSE linkage processing

PSE linkage processing controls the PSE linkage by using the H-7338 protocol.

■ Host computer linkage processing

Host computer linkage processing controls the host computer linkage by using the H-7338 protocol.

■ CPU-to-CPU linkage

CPU-to-CPU linkage transfers the contents of global area (PI/O G area) from the specified address to another CPU (2α, 2αE, 2αH, 2αHf) as much as the specified number of words.

1 GENERAL DESCRIPTION

■ CPU-to-CPU PSE linkage

CPU-to-CPU PSE linkage links one PSE to two or more CPUs (2α, 2αE, 2αH, 2αHf).

■ External device linkage

External device linkage provides the RS-422 interface for linking external devices (personal computers, CRT, T/W, etc.). A protocol must be selected according to the external device.

■ Task management

Task management manages task start, task restart, and idle and dormant state of tasks. It also schedules tasks when tasks are started or restarted.

■ Timer management

Timer management sets, cancels, and delays cyclic start.

■ Factor management

Factor management sets task start factor and resets it after a task is fetched.

■ State control

State control modifies the execution level (hardware level) and task level.

■ Resource management

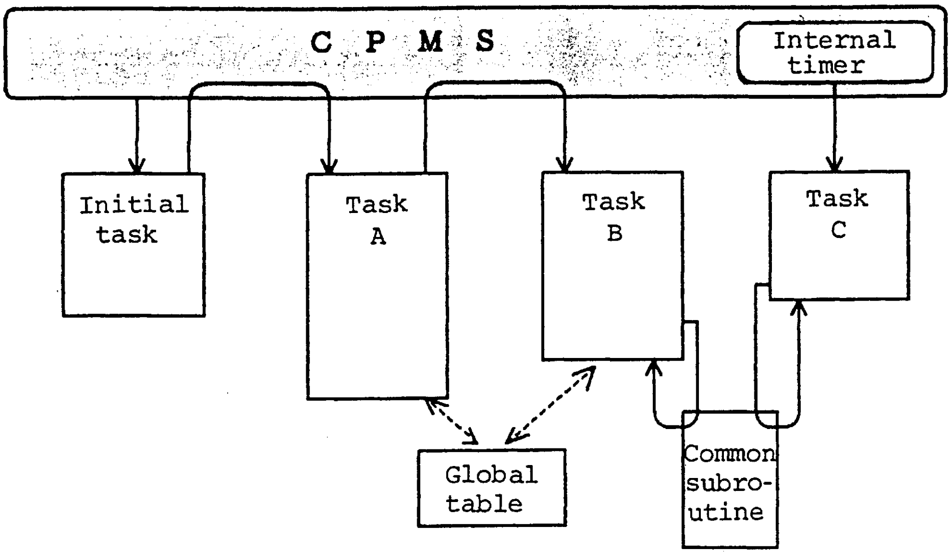
Resource management exclusively controls user-defined resources shared by tasks.

2 TASK MANAGEMENT

2 TASK MANAGEMENT

2.1 Task Configuration

A user created program is constructed as shown below:



■ Task number

A task has a task number (TN) for identification.

P coil and task classification

P coil number	Task number	Classification	Start method	Explanation
P001	1	Initial task	Started when CPU power supply is turned on or CPU is reset.	When the CPU power supply is turned on, the initial task is started. A task that initializes the system is allocated as the initial task.
P002 ~ P07F	2 ~ 127	User task	Started when the P coil is excited or a macro instruction (queue) is issued.	A control task created by the user is allocated as a user task.
P080	128	System task	Started by the system OS.	A system task (such as debugger task) is allocated as the system task. The user must not use this task.

● For 2α, 2αE (2αH, 2αHf) tasks can be started (queued) when the P coil is excited by a sequence program. The P coil number is equal to the task number.

- Global table

The global table is used to transfer data between tasks.

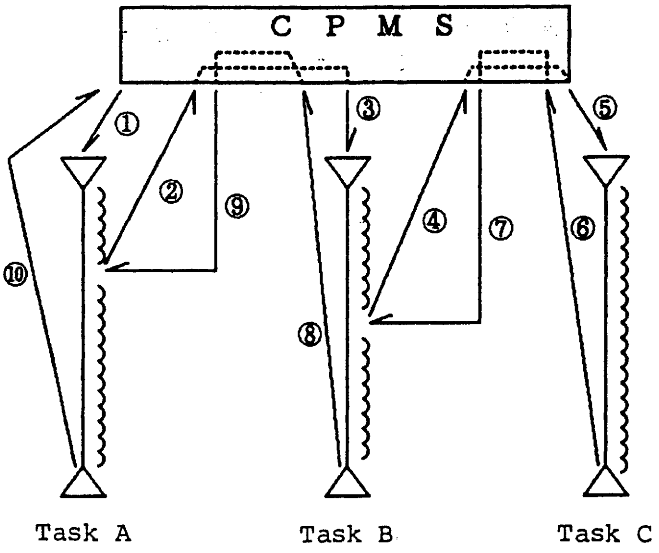
- Common subroutines

The common subroutines are reentrant subroutines that can be used by task.

2.1.1 Operations

User-created tasks are managed by CPMS for start, interrupt, and terminate operations. Each task has a priority level. When tasks are started, tasks are processed beginning from the one that has the highest priority level. For example, if a request to start a task that has a priority level higher than the current task is issued, the current task execution is interrupted and the task with higher level is executed. The interrupted task is made to wait until execution of the task with higher priority level ends.

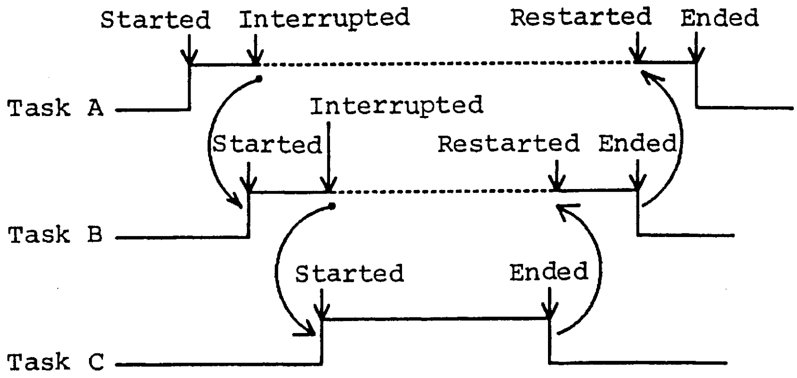
- Operation example



● Task level: C>B>A

● Numbers: Execution order

- Timing chart

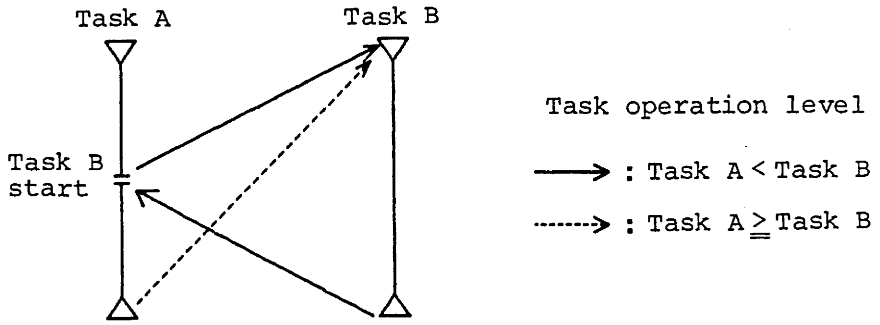


2.1.2 Linkage between CPMS and tasks

CPMS provides various instructions related to task start, end, and cancel. These instructions are used by user tasks to request CPMS processing. These instructions are called macro instructions The user uses these macro instructions to realize multitasking.

◆◆ Task start operation examples using macro instructions ◆◆

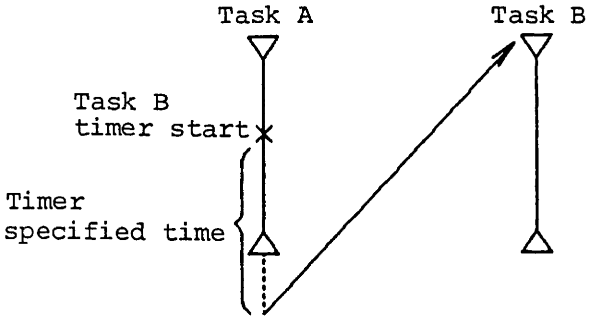
● QUEUE macro instruction



If the level of task B is higher than task A, control is passed to task B when the request to start task B is isued. If the level of task B is equal to or less than the level of task A, control is passed after task A execution ends.

If a request to start a task whose level is equal to or higher than the level of task B has been issued before control is passed to task B, the task is executed first then control is passed to task B.

● TIMER macro instruction



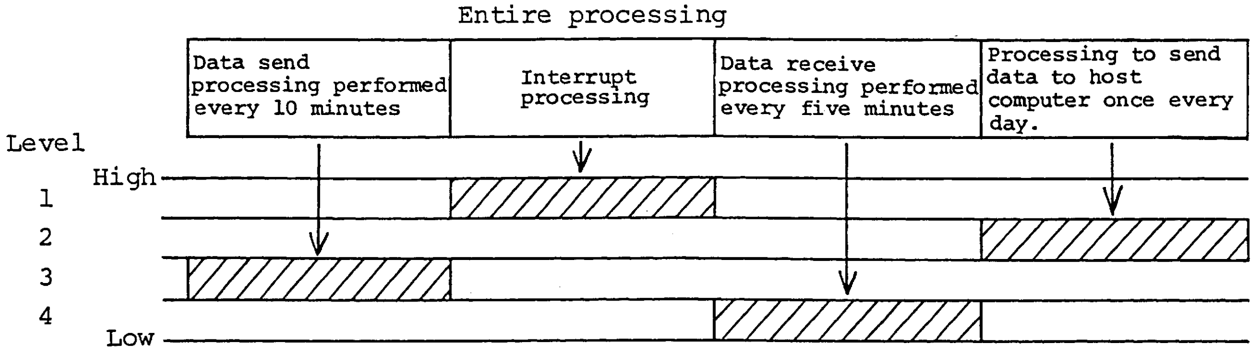
A request to start task B is issued after the timer specified time elapses. If a request to start a task whose level is equal to or higher than the level of task B has been issued, the task is executed first then control is passed to task B.

2.1.3 Task levels

Each task has a priority level. When an attempt is made to execute two or more tasks, the tasks are executed according to the priority levels beginning from the task that has the highest priority level. If two tasks are in the same priority level, the one for which a start request is issued first is executed first. There are five priority levels from 0 to 4. The lower the number is, the higher the priority. A priority level is allocated when the debugger registers a task.

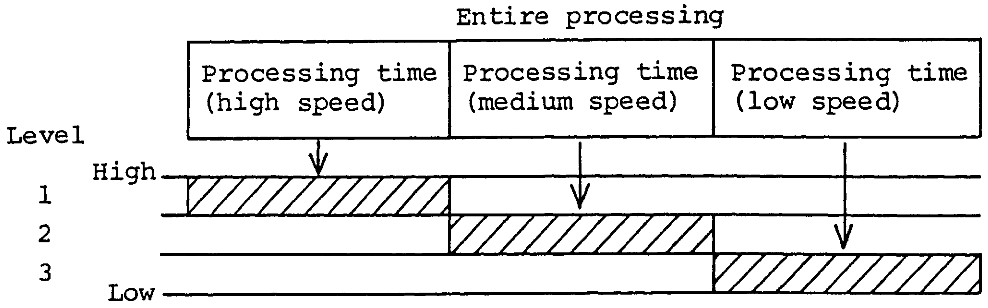
Determination of task operation levels differs depending on the system configuration.

■ Determination of task operation levels according to processing


Entire processing
Data send processing performed every 10 minutes
Interrupt processing
Data receive processing performed every five minutes
Processing to send data to host computer once every day.
Level
High
1
2
3
4
Low

Interrupts are regarded as the first priority and the highest level is given to interrupt processing. Then, levels are determined for processing that is performed less often.

■ Determination of task operation levels according to processing time


Entire processing
Processing time (high speed)
Processing time (medium speed)
Processing time (low speed)
Level
High
1
2
3
Low

Because tasks with high-speed processing are made to wait for a long time if higher operation levels are given to tasks that has low-speed processing, higher levels are given to tasks that have high-speed processing.

2.1.4 Subroutines

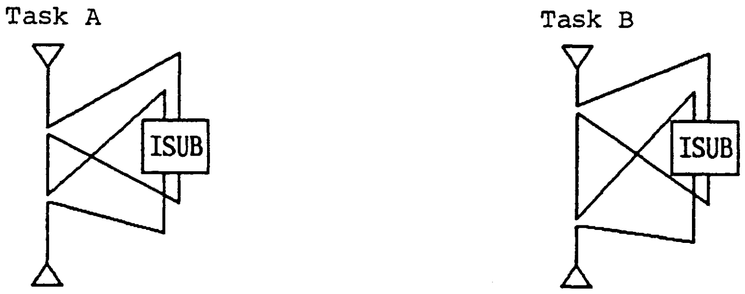
Two types of subroutine are provided. ISUB is used by a specific task and RSUB is used by two or more tasks.

ISUB is directly linked to the main task (or a subroutine) that calls the ISUB.

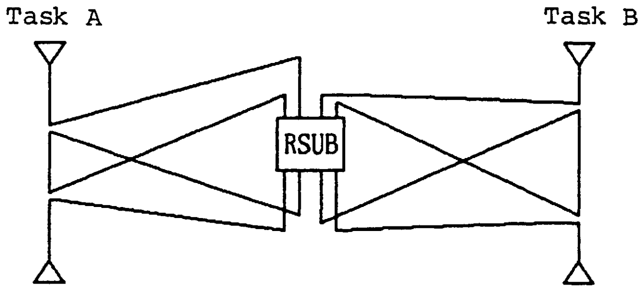
RSUB is stored in a fixed address and operation jumps from each task to this address.

A library is linked as an ISUB.

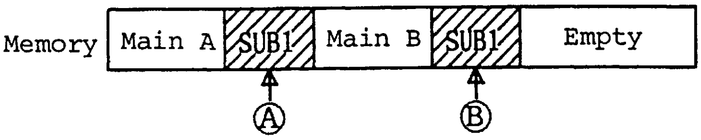
ISUB: Internal subroutine



RSUB: Resident subroutine

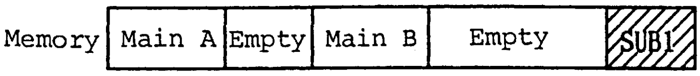


■ Differences in memory
[For SUB1 defined as ISUB]



The same subroutine (SUB1) is stored in two areas Ⓐ and Ⓑ and memory is used wastefully.

[For SUB1 defined as RSUB]



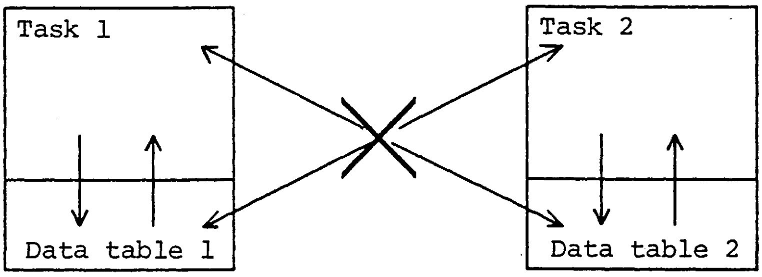
SUB1 is made resident so that memory has sufficient empty areas.

2.1.5 Data tables

Data tables are classified into two types, internal task data tables and global data tables.

[Internal task data table]

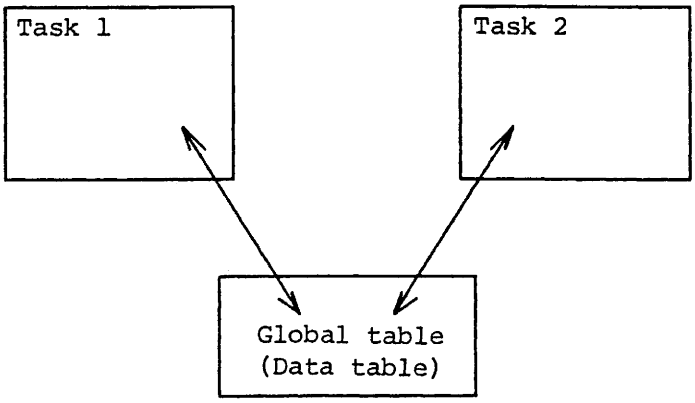
An internal task data table is used by only one task.



Task 1 can use only the data table 1. Task 1 cannot read/write the data table 2. Task 2 can use only the data table 2. Task 2 cannot read/write the data table 1.

[Global table]

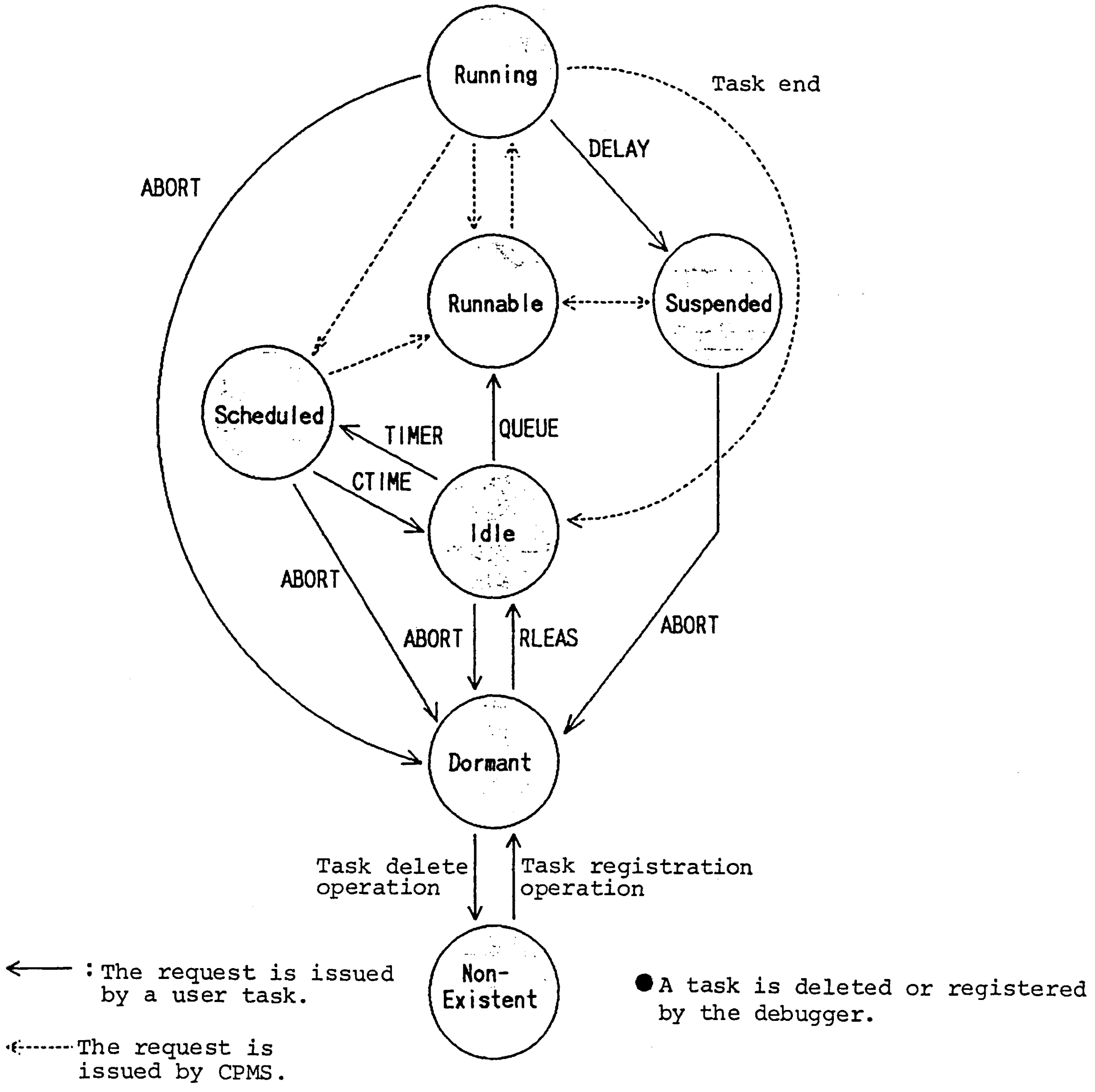
A global table is used by two or more tasks. Data is read from or written to this global table by two or more tasks. Using a global table, data can be transferred between tasks.



Both task 1 and task 2 can read data from or write data to the global table (data table). Using the global table, data can be transferred between task 1 and task 2.

2.2 Task States

A task may be in execution or interrupt state.


Running
Task end
DELAY
ABORT
Runnable
Suspended
Scheduled
TIMER
QUEUE
CTIME
Idle
ABORT
ABORT
RLEAS
ABORT
Dormant
Task delete operation
Task registration operation
Non-Existent
←── : The request is issued by a user task.
←------ The request is issued by CPMS.
● A task is deleted or registered by the debugger.

■ Non-existent

The task is not registered to CPMS.

■ Dormant

Starting the task is inhibited.

■ Idle

A request to start the task can be accepted.

■ Runnable

A request to start the task has been accepted and the task is waiting to be executed.

■ Running

The task is being executed.

■ Scheduled

A request to start the task can be accepted and the task will automatically enter runnable state after a fixed time.

■ Suspended

Task execution has been interrupted and the task is waiting for an event to occur.

◆◆ Task states and macro instructions ◆◆

A macro instruction is used to change the state of a task. The state of a task changes as shown below.

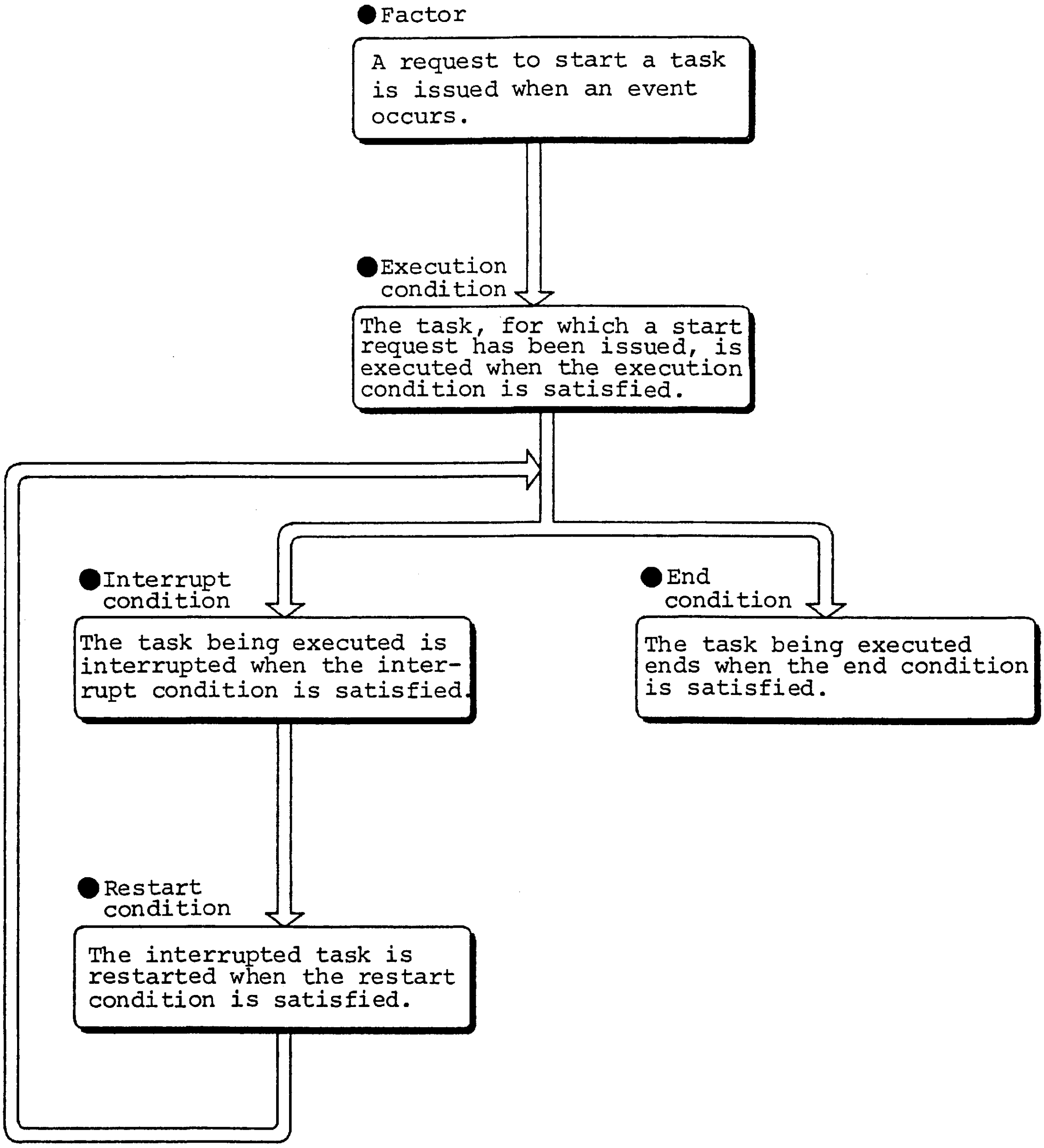
Macro name	Task which issues the marco instruction		Task for which the macro instruction is issued.	
	Before	After	Before	After
RLEAS	Running	Running	Dormant	Idle
QUEUE	Running	Running	Idle	Runnable
ABORT	Running	Running	Optional	Dormant
TIMER	Running	Running	Idle	Scheduled
CTIME	Running	Running	Scheduled	Idle
DELAY	Running	Suspended	———	———
CHAP	Running	Running	Running	Runnable

CAUTION

The STOP SWITCH on the CPU module does not affect the operation of CPMS task. Outputs can still be controlled by CPMS task or by the action of programmers using these tasks while the switch is in the "STOP" position.

2.3 Task Operations

Task execution flows as follows after it is started until it ends:



A request to start a task is issued when an event occurs.

The task, for which a start request has been issued, is executed when the execution condition is satisfied.

The task being executed is interrupted when the interrupt condition is satisfied.

The task being executed ends when the end condition is satisfied.

The interrupted task is restarted when the restart condition is satisfied.

◆◆ Factor ◆◆

Event	Explanation
A QUEUE macro instruction is issued.	It is requested to start the task specified by the QUEUE macro instruction parameter.

◆◆ Execution condition ◆◆

Condition	Explanation
There is no task whose priority level is higher than the local task, or whose priority level is equal to the local task priority level but a request to start the task has been issued before the request to start the local task. Further, the local task is in memory and has been registered to CPMS.	The local task is executed when it has the highest priority among the tasks in the queue block and the local task has been generated.

◆◆ Interrupt condition ◆◆

Condition	Explanation
A resource required for task execution cannot be used.	If one of the resources (such as CPU) required for task execution is being used by another task, execution of the local task is interrupted and the task is made to wait.
A task with higher priority is in executable status.	By an interrupt, the task with higher priority is started and control is passed to the higher priority task if it is in executable status.
The local task interrupts its own execution.	If the local task interrupts itself in order to have synchronization, control is passed to another task.

◆◆ End conditions ◆◆

Condition	Explanation
Task execution ends.	The task is removed from the execution wait queue.
An ABORT macro instruction is issued.	Task execution is terminated if an ABORT instruction is issued to the task.
Processing enters status in which it can no longer be continued.	CPMS automatically aborts the erroneous task.

◆◆ Restart condition ◆◆

Condition	Explanation
A resource required for task execution becomes usable.	A resource required for task execution is freed by another task and becomes available.
The cause of interrupt is removed.	The cause of interrupt (delay) is removed.
All the tasks with higher priority levels are interrupted or ended.	The local task is not restarted until all tasks which have higher priority than the local task become inoperable.

3 SYSTEM MANAGEMENT

3 SYSTEM MANAGEMENT

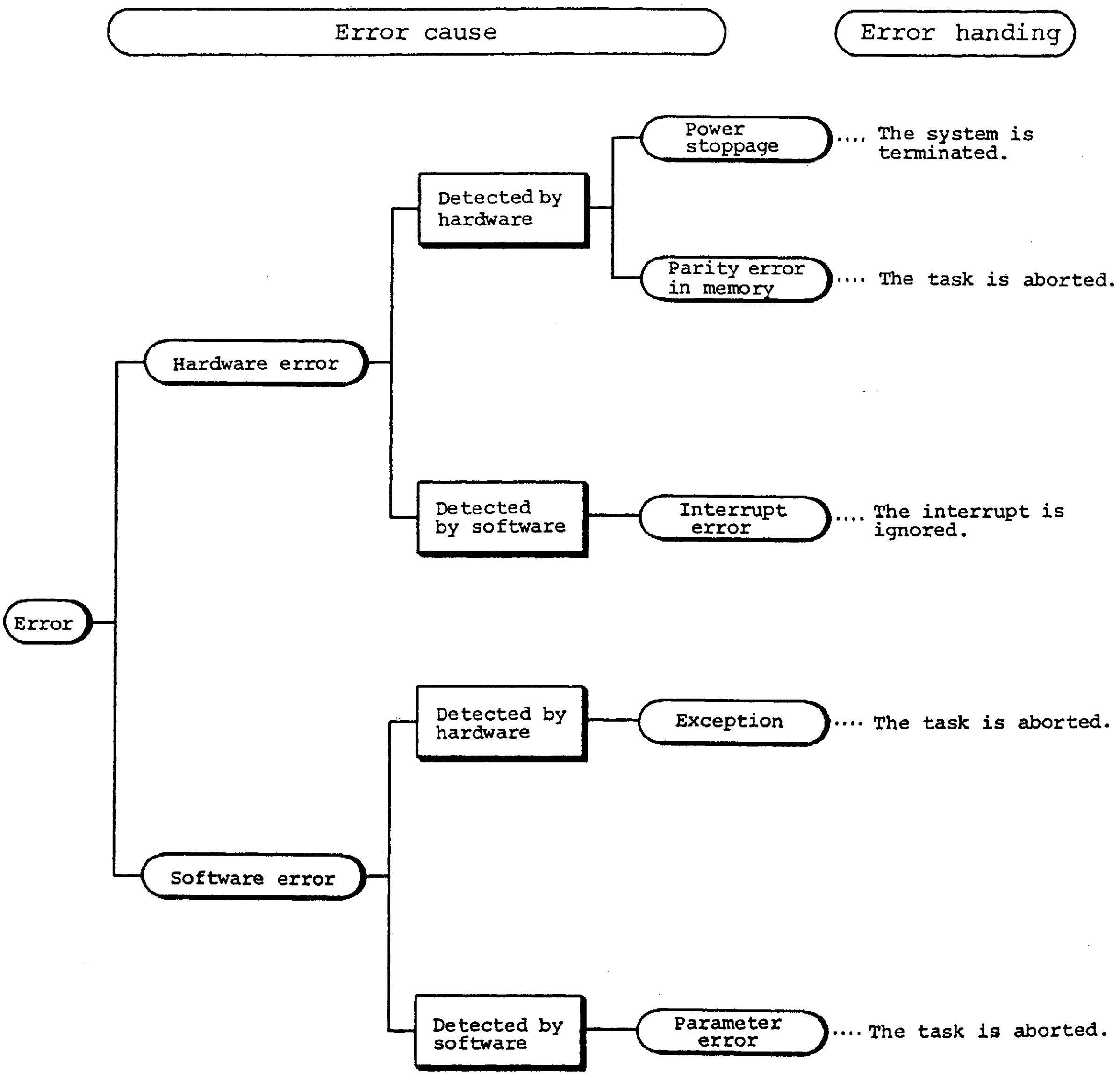
3.1 Initiating the System

When the power supply is turned on, CPMS starts the initial task. The initial task is a user task to be started first. All user tasks other than the initial task has dormant status when the power supply is turned on. Therefore, the initial task has to issue RLEAS macro instructions for the other user tasks required by the job so that the tasks can accept the start requests. The task number of initial task is 1.

3.2 Error Handling

The operating system of 2α series has the improved RAS functions for the higher system reliability. The operating system checks hardware-level and software-level error and performs retry operation for some software. If an error occurs for which the operating system cannot perform retry operation, the operating system collects the information of the error, links error handling to user-installed subroutines, then terminates the task which has issues the request.

◆◆ Error causes and error handling ◆◆



3.3 Installing Subroutines

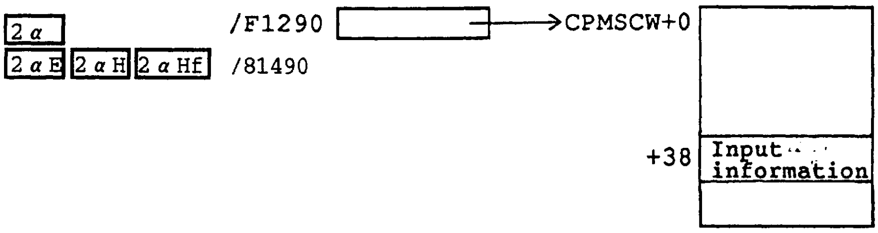
If an event (error) occurs in the system, the operating system performs error handling. The user can incorporate error handling in the system and register it as a subroutine. One user-installed subroutine can be registered for one event.

3.3.1 Types

The following subroutines can be incorporated in user programs. The linking of these built-in subroutines do not require register and save restore operations within the subroutines. This allows the subroutines to be used in C language programs.

Subroutine		Operation Level	Applicable Event	Available Information	Issuable Macro Instr.	Remarks
Name	Abbr.					
System Down Subroutine	SDS	6	A hardware or program error during OS execution	Address of Error Freeze Area (ERSTK)	None	
CPU Error Subroutine	CPES	6	A hardware or program error during a task execution	Address of Error Freeze Area (ERSTK)	queue rleas	
Exit Subroutine	EXS	5	End of a task	TN of the ended task	queue rleas	After release of resource
Abort Subroutine	ABS	5	Issue of Abort macro instruction	TN of a task being aborted	queue rleas	Ditto
Parameter Check Subroutine	PCKS	5	A macro parameter error	Address of Error Freeze Area (SVCEB)	queue rleas	

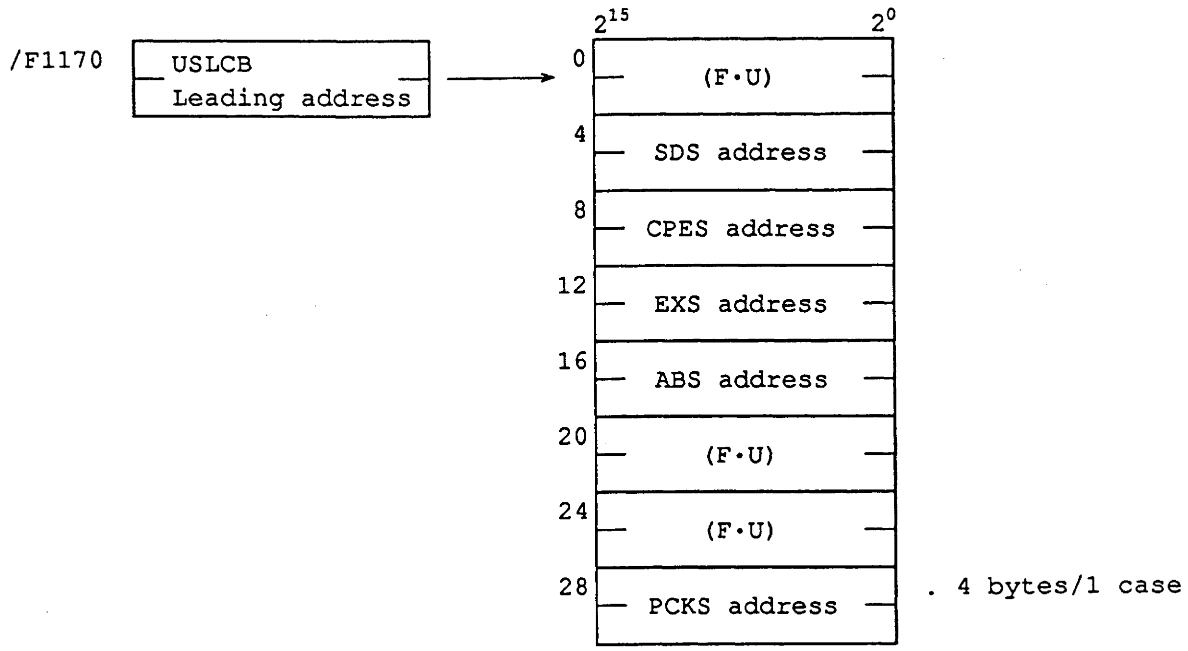
- CPMSCW is located at address /F1290 for the 2α, or at address /81490 for the 2αE, 2αH, and 2αHf.
- The input data is stored as the 38th long word counting from CPMSCW.



3.3.2 Registration

A subroutine is installed by the user writing its address at the corresponding area in the user USLCB (Subroutine Link Control Block). The start address of USLCB is stored as a long word at address /F1170 (SEQCB Sequence Control Block). Be careful to install the subroutine correctly.

▷▷ USLCB ◁◁



3.3.3 Input information

The error information input from the built-in subroutines include the following two tables:

- ERSTK: Error Stack
  Edited at system down or when a CPU error occurs.

- SVCEB: Supervisory Call Error Block
  Edited when a macro parameter error occurs.

2 α

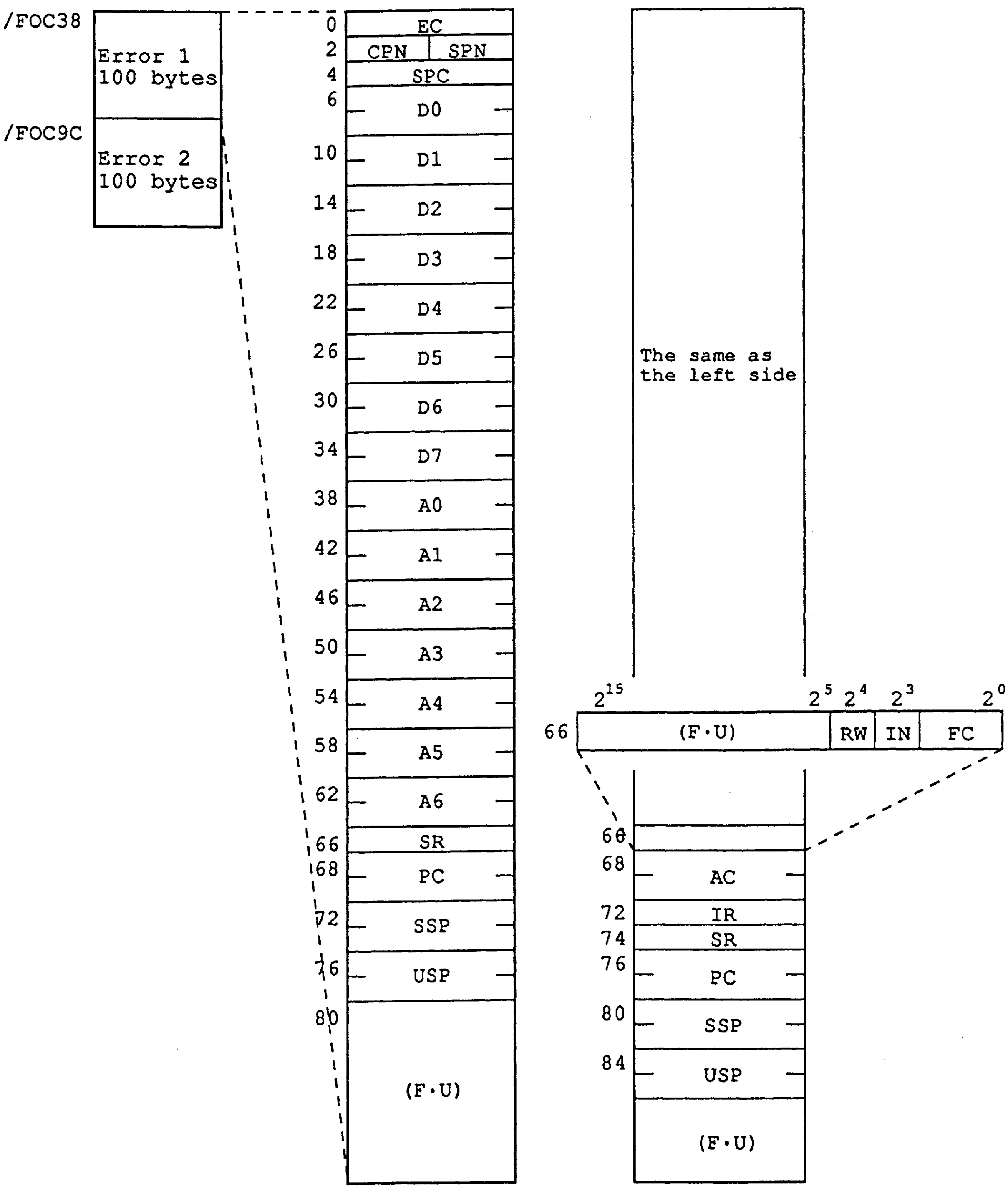
3.4 Error Log

●● Structure of ERSTK Table for S10/2α ●●

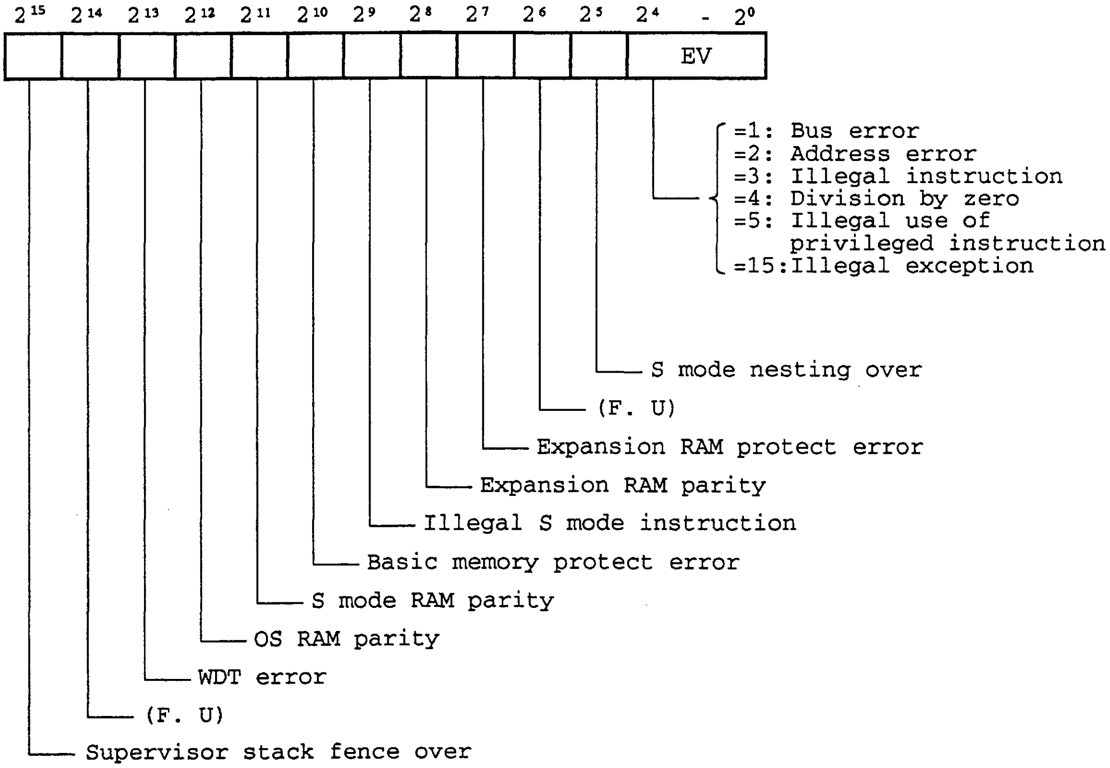
This table stores data for two errors if the two errors occur in succession. Different data is stored if EC in the indicates an address error.

▷▷ ERSTK ◁◁

▷▷ ERSTK (When address error occurs:) ◁◁



EC: Error Code



CPN : Task (P coil) No.

SPN : S mode program No.

SPC : S mode program counter (Valid only for illegal S mode instruction or S mode RAM parity)

D0-7: Contents of data register when an error occurs

A0-6: Contents of address register when an error occurs

SR : Contents of status register when an error occurs

PC : Program Counter (MPU)

SSP : System Stack Pointer

USP : User Stack Pointer

RW : Read operation (=1) and Write operation (=0)

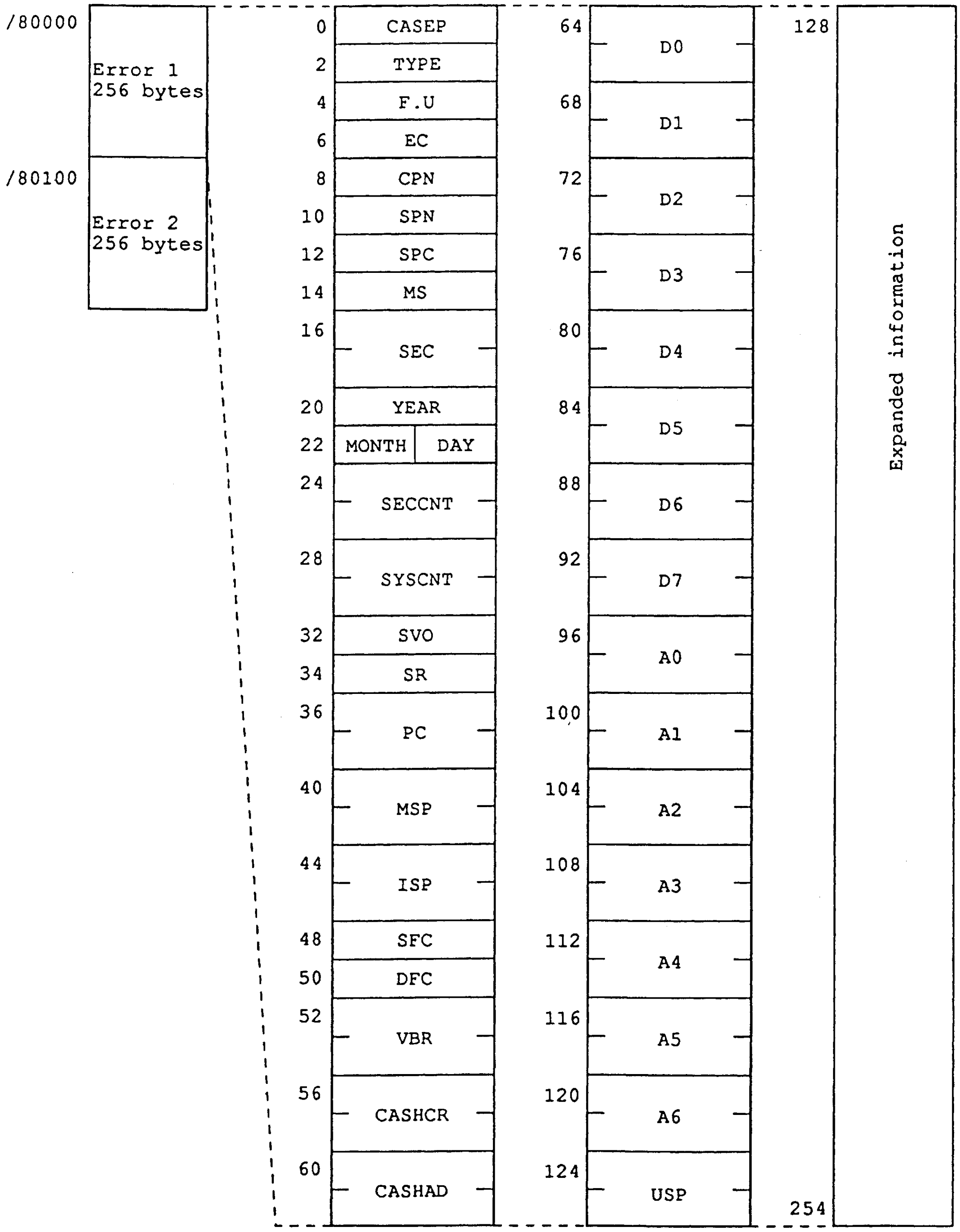
IN : Instruction (=0), Others (=1)

```
FC  :  Function Code  ⎧ =1 ... user data
                      ⎪ =2 ... user program
                      ⎨ =5 ... supervisor data
                      ⎩ =6 ... supervisor program
AC  :  Access Address
IR  :  Instruction Register (Instruction when an error occurs)
F•U :  Reserved for future use
```

2αE
2αH
2αHf

•• Structure of ERSTK Table for S10/2αE ••

This table stores data for two errors if the two errors occur in succession. The "Expanded Information" in the table differs depending on the contents of "stack frame format".

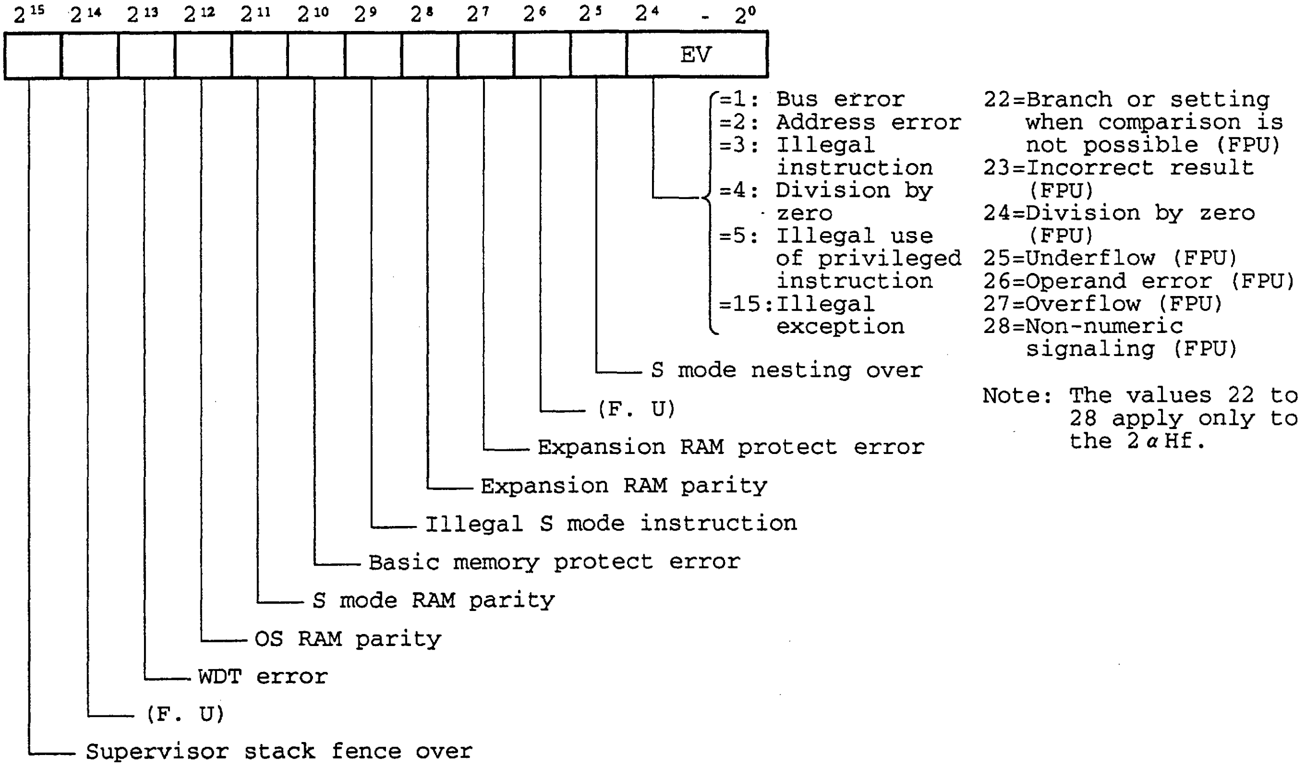


/80000	Error 1 256 bytes
/80100	Error 2 256 bytes

0	CASEP	64	D0	128	Expanded information
2	TYPE				
4	F.U	68	D1		
6	EC				
8	CPN	72	D2		
10	SPN				
12	SPC	76	D3		
14	MS				
16	SEC	80	D4		
20	YEAR	84	D5		
22	MONTH DAY				
24	SECCNT	88	D6		
28	SYSCNT	92	D7		
32	SVO	96	A0		
34	SR				
36	PC	100	A1		
40	MSP	104	A2		
44	ISP	108	A3		
48	SFC	112	A4		
50	DFC				
52	VBR	116	A5		
56	CASHCR	120	A6		
60	CASHAD	124	USP	254	

2αE
2αH
2αHf

EC: Error Code



EV:
=1: Bus error
=2: Address error
=3: Illegal instruction
=4: Division by zero
=5: Illegal use of privileged instruction
=15: Illegal exception
22=Branch or setting when comparison is not possible (FPU)
23=Incorrect result (FPU)
24=Division by zero (FPU)
25=Underflow (FPU)
26=Operand error (FPU)
27=Overflow (FPU)
28=Non-numeric signaling (FPU)

Note: The values 22 to 28 apply only to the 2αHf.

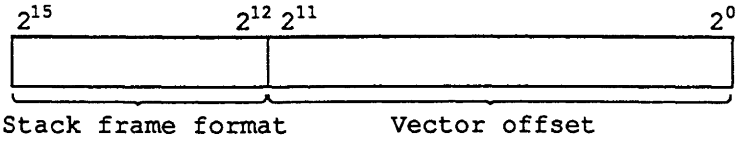
2^{5}: S mode nesting over
2^{6}: (F. U)
2^{7}: Expansion RAM protect error
2^{8}: Expansion RAM parity
2^{9}: Illegal S mode instruction
2^{10}: Basic memory protect error
2^{11}: S mode RAM parity
2^{12}: OS RAM parity
2^{13}: WDT error
2^{14}: (F. U)
2^{15}: Supervisor stack fence over

CASEP : Case point
TYPE : RAM - OS time=/6820, ROM - OS time=/6821
EC : Error code
CPN : Task (P coil) No.
SPN : S mode program No.
SPC : S mode program counter
(Effective only for illegal S mode instruction or S mode RAM parity)
MS : Millisecond
SEC : Second
YEAR : Year
MONTH : Month
DAY : Day
} Are set only when the memory with clock feature is mounted.
SECCNT: Seconds counter

2αE
2αH
2αHf

SYSCNT	:	System counter
SVO	:	Stack frame format + Vector offset
SR	:	Contents of status register
PC	:	Program counter (MPU)
MSP	:	Master stack pointer
ISP	:	Interrupt stack pointer
SFC	:	Source function code
DFC	:	Destination function code
VBR	:	Vector base register
CASHCR	:	Cache control register
CASHAR	:	Cache address register
D0-D7	:	Contents of data register
A0-A6	:	Contents of address register
USP	:	User stack pointer

SVO: Stack Frame Format + Vector Offset



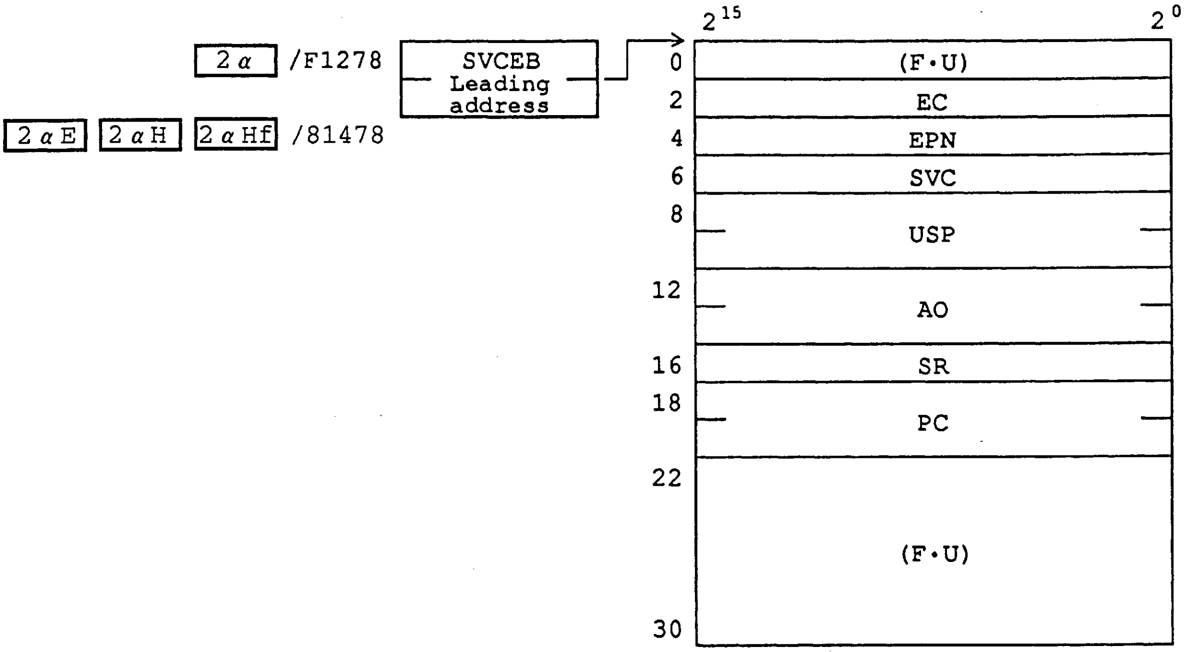
Stack frame format and expanded information
There is no expanded information when the stack frame format is other than 2, 9, A, or B.

2 α E
2 α H
2 α Hf

	2	9	A	B	
128	INS_A	INS_A	IR	IR	128
			SSW	SSW	130
132		IR	IPS_C	IPS_C	132
		IR	IPS_B	IPS_B	134
136		IR	DCFA	DCFA	136
		IR			
140			IR	IR	140
			IR	IR	142
144			DOB	DOB	144
148			IR	IR	148
			IR	IR	150
152				IR	152
				IR	154
				SB_A	156
				IR	160
				IR	162
				DIB	164
				IR 22 words	168
					210

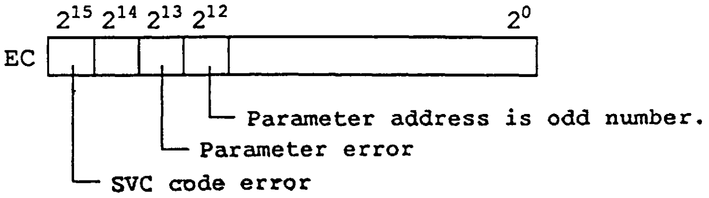
INS_A: Instruction address
IR : Internal register
SSW : Special status word
IPS_C: Instruction pipe stage_C
IPS_B: Instruction pipe stage_B
DCFA : Data cycle fault address
DOB : Data output buffer
SB_A : Stage_B address
DIB : Data input buffer

●● Structure of SVCEB Table ●●



0	(F•U)
2	EC
4	EPN
6	SVC
8	USP
12	A0
16	SR
18	PC
22	(F•U)
30	

EC : Error Code

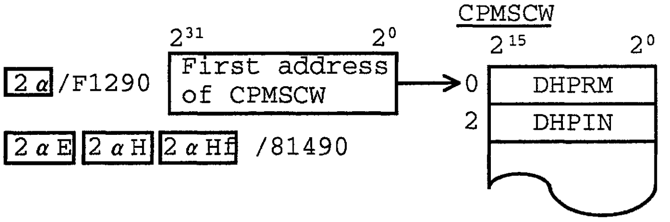


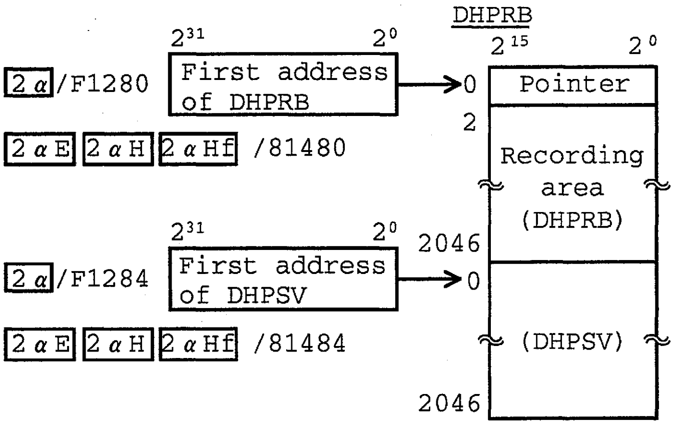
EPN: Program No. which contains an error
SVC: SVC macro No.
USP: USP when an error occurred
A0 : A0 register when an error occurred
SR : SR when a macro instruction is issued
PC : Return address of macro instruction
F•U: Reserved for future use

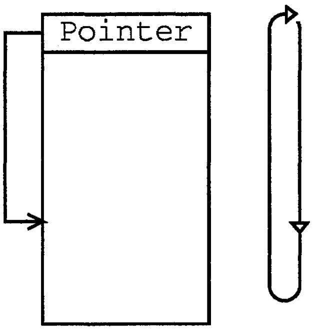
3.5 DHP

The debugging helper (DHP) tool traces status changes of the operating system. Should the operating system cause a problem, DHP is used to check system conditions before the problem occurred.

DHP tables are shown below.







DHP entries are recorded sequentially.

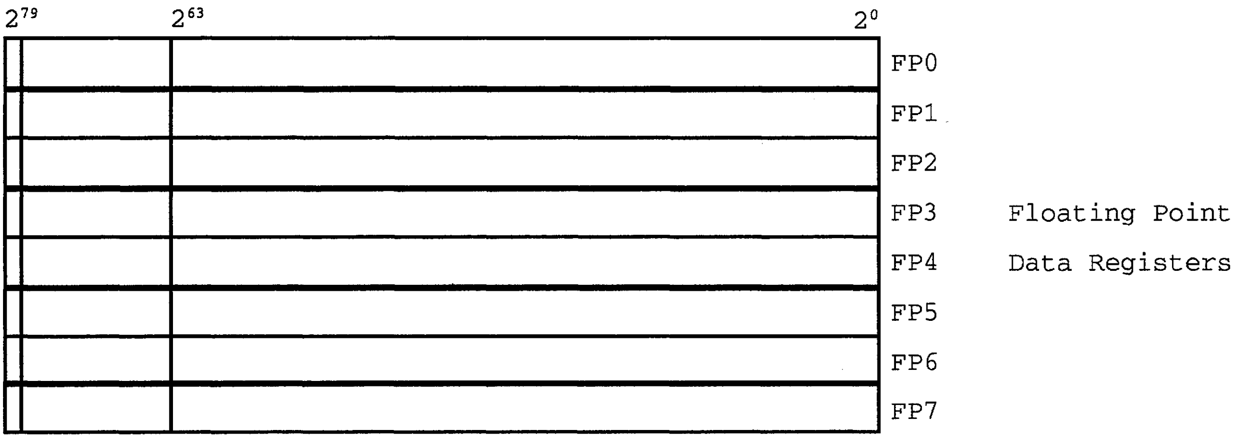
- DHPRM (DHP recording mode)
  - 2: All recording mode
  - 1: CPMS macro recording mode
  - Other: No DHP entry is recorded.
- DHPIN (DHP initial mode)
  - 1: On resetting (GR issued), DHPRB and DHPSV are cleared (zero).
  - Other: On resetting (GR), DHPRB and DHPSV are not cleared.
- DHPRB (DHP recording area)
  The DHP trace buffer is 2048 bytes deep. The first word is a pointer to the next record.
- DHPSV (DHP save area)
  Area to which the contents of DHPRB are saved if:
  1: The CPU goes down.
  2: The CPU causes an error.

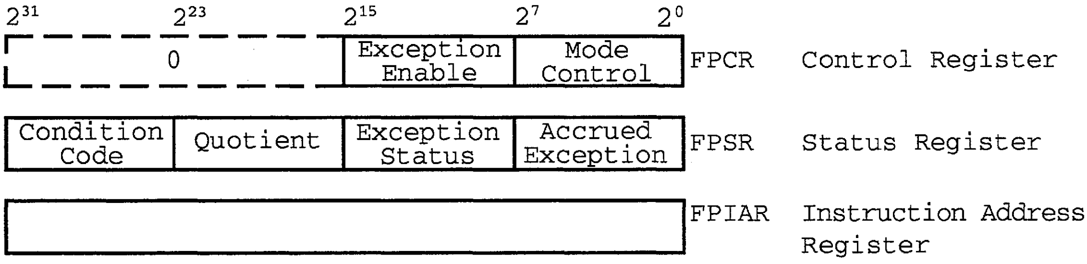
3.6 Floating-Point Operation

Only the H-S10/2αHf of the 2α series supports floating-point operation. Floating-point operation conforms to the IEEE standard. For details on floating-point registers and floating-point functions, refer to the "CP/M-68K V1.2 Operating System Guide" and commercially available manuals for compilers.

The following paragraphs describe floating-point registers and the error log for floating-point operations.

<Floating-point registers>





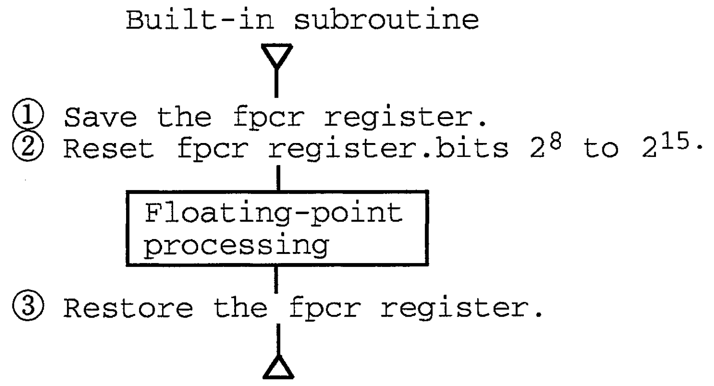
3.6.1 Notes on use of floating-point operations

When using built-in subroutines to perform floating-point operations, suppress exception traps to prevent system errors. To suppress exception traps, reset bits 2^8 to 2^{15} (ø)(of the control register (cr) in the floating-point coprocessor (fpcp) at the beginning of the built-in subroutine. To do this, the cr of the fpcp (called the fpcr) must be saved and rewritten before the floating-point operation is processed, the fpcr must be restored after floating point processing as shown below.

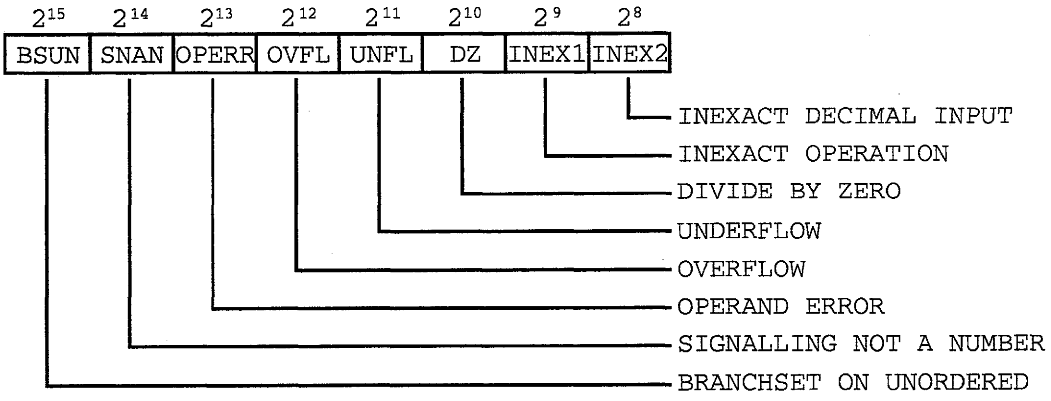
A sample assembly language routine for saving, rewriting, and restoring the cr of the fpcp is also shown below.

C does not support library functions to rewrite the cr of the fpcp. For details of library functions, refer to the "CRM-68K V1.2 Operating System Guide."

<Processing with the fpcp cr in a built-in subroutine and cr configuration>


Built-in subroutine
① Save the fpcr register.
② Reset fpcr register.bits 2^8 to 2^15.
Floating-point processing
③ Restore the fpcr register.

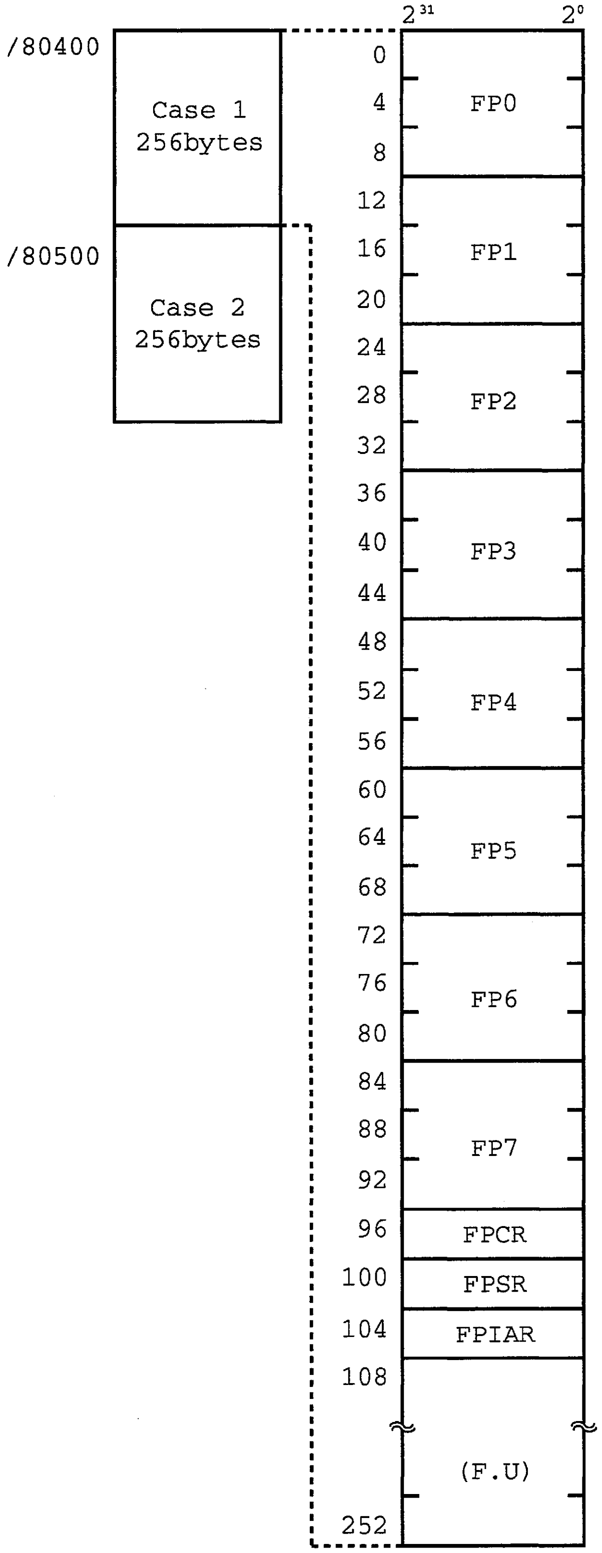
<Configuration of the fpcp cr>



<Saving, rewriting, and restoring the fpcp cr>

	Coding
Assembler	⋮ fmove.1 fpcr, (Save area) ··· ① fand.1 0xff,fpcr ············ ② ⋮ (Floating-point processing) ⋮ fmove.1 (Save area),fpcr ··· ③ ⋮

3.6.2 Error log for floating-point operation



4 MACRO INSTRUCTION

4 MACRO INSTRUCTION

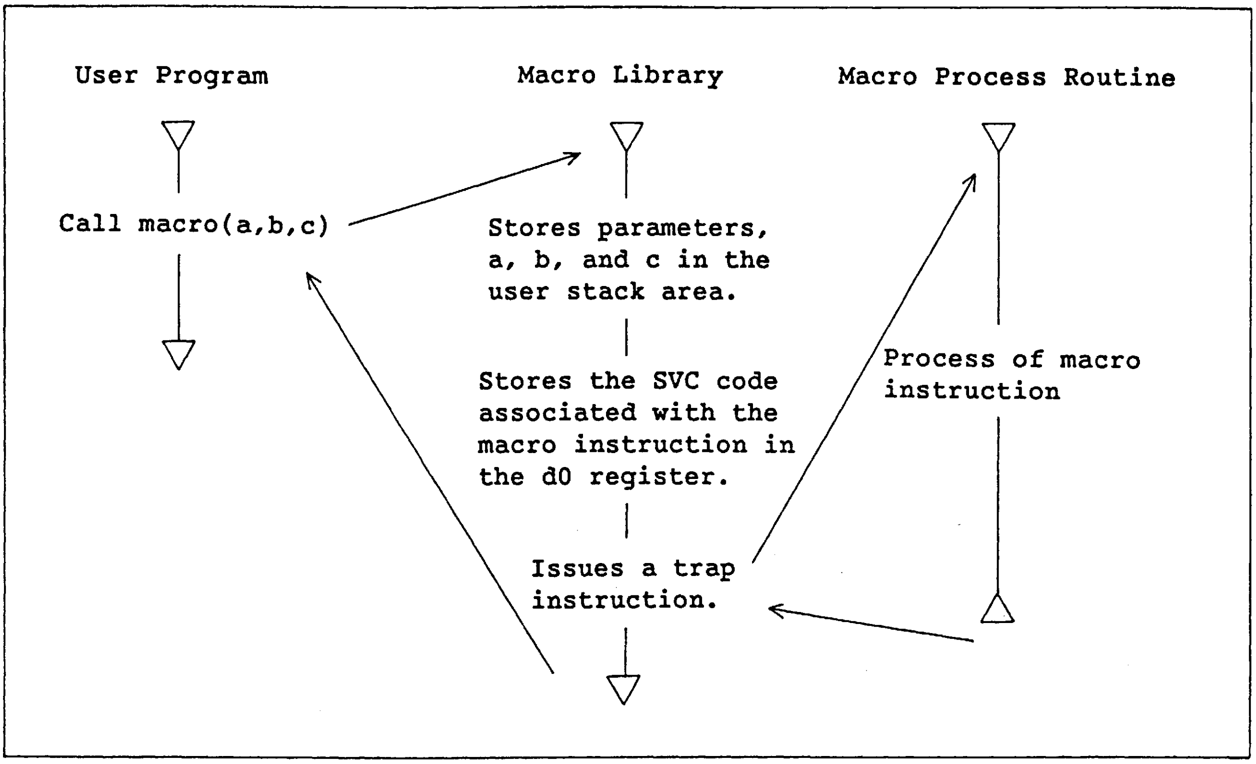
4.1 Macro Instruction

The macro instruction is an instruction issued by the user program (task) to request processing by the operating system (OS) (Compact PMS). When a macro call is used in a user program the macro instruction is expanded to a TRAP (software interrupt instruction) contained in the macro library.

When the program is executed, this trap instruction causes a hardware interrupt, passing program control to OS, and the OS initiates the macro process routine. (The trap instruction changes the user mode to the supervisor mode, causing an interrupt by software.) The user mode is a mode in which a user program is executed. The supervisor mode is the system task process mode for OS processing.

4.1.1 Macro library

The macro library is a group of subroutines for expanding macro instructions written in a high-level language to the trap instruction. When called, it stores the parameters (arguments) in the user stack area in the order defined for each macro instruction, and then issues the trap instruction. When writing in C language, the programmer can write an instruction on the macro instruction level without taking the trap instruction into consideration, but must be concerned with the trap instruction if it is written in Assembly language.


User Program
Macro Library
Macro Process Routine
Call macro(a,b,c)
Stores parameters, a, b, and c in the user stack area.
Stores the SVC code associated with the macro instruction in the d0 register.
Issues a trap instruction.
Process of macro instruction

Provided macro libraries differ in their storage location according to the development environments.

- Development on the PSEα : Macro libraries are provided in the CPMSMLIB file on the CPMS system floppy disk.

- Development on a personal computer: Macro libraries are provided in the CPMS. LIB file on the system floppy disk of the C Program Development Environment System (RPDP/S10).

4.1.2 General rules

◆◆Passing Parameters ◆◆

When the macro library is used, any parameter is passed using its address.

■ When prepared in C language:

```
long tn ;
   ⋮
tn=100 :
   ⋮
macro(&tn) ;
   ⋮
```

Describe the address where tn(=100) is stored as an argument as shown on the left side. &tn is the pointer to tn, and indicates the address storing tn. Be careful not to write macro(tn).

Many other descriptions are permitted in the C language. What is important is that the address of a parameter is passed to the macro library correctly. The following descriptions produce the same results. Use any method convenient for your situation. The macro descriptions shown in Chapter 2 are only examples. Other descriptions may be used if they produce the same result.

● When a parameter is the whole array:

```
      ⋮
long X[n] ;
      ⋮
macro(X) ;
      ⋮
```

- When a parameter is one element of an array (The following three descriptions are equivalent.):

```
      ⋮
long X[n] ;
      ⋮
 X[i] =100 ;
      ⋮
macro(&X[i]);
      ⋮
```

```
      ⋮
long X[n] ;
      ⋮
 X[i] =100 ;
      ⋮
macro(X+i) ;
      ⋮
```

```
      ⋮
long *X[n] ;
      ⋮
*  X[i] =100 ;
      ⋮
macro(X [i] ) ;
      ⋮
```

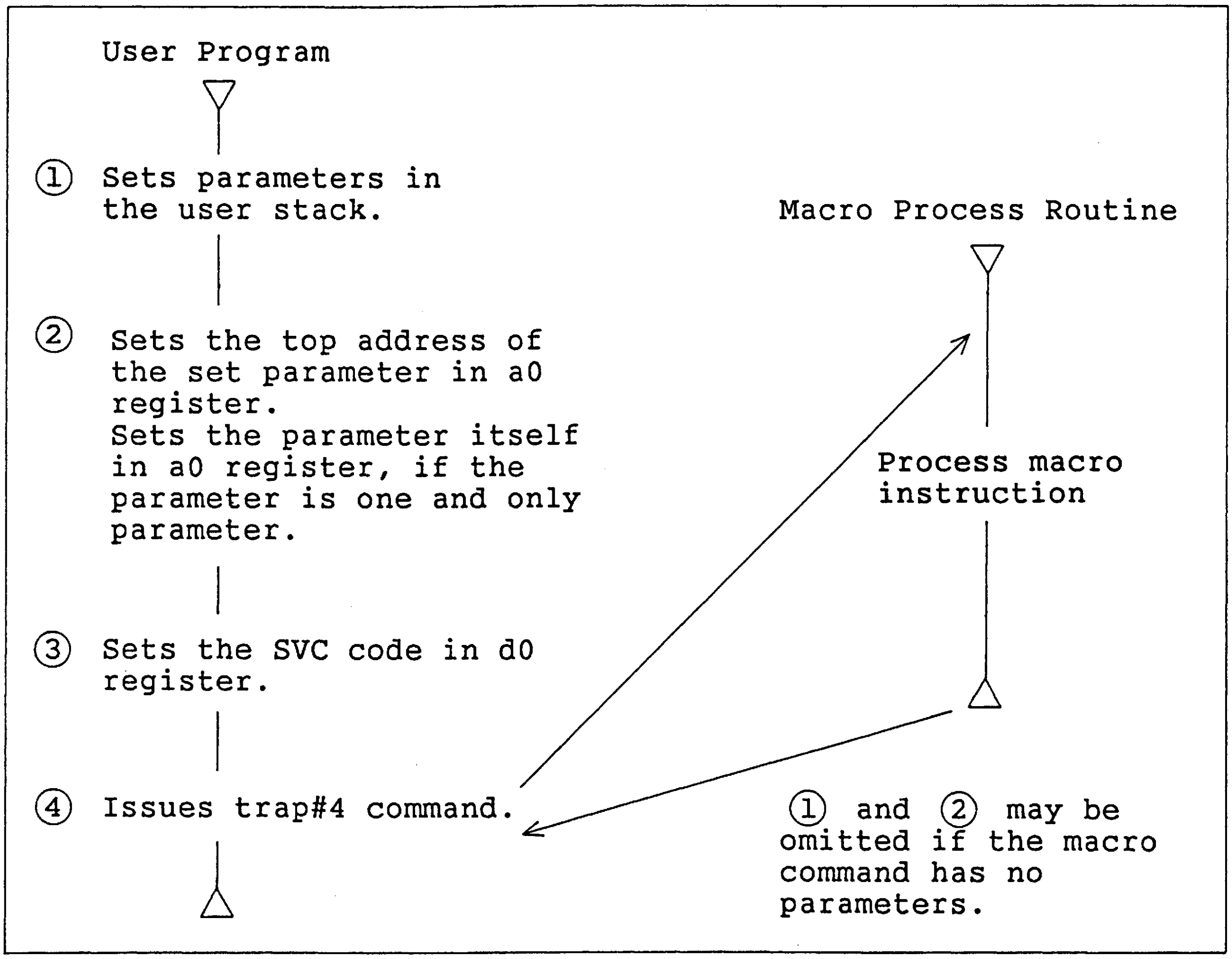
- When a parameter is a simple variable (The following two descriptions are equivalent.):

```
      ⋮
long X ;
      ⋮
X=100 ;
      ⋮
macro(&X) ;
      ⋮
```

```
      ⋮
long *X ;
      ⋮
*X=100 ;
      ⋮
macro(X) ;
      ⋮
```

■ When prepared in assembler:

Issue the trap #4 command after storing the parameters in the user stack.


User Program
① Sets parameters in the user stack.
② Sets the top address of the set parameter in a0 register.
Sets the parameter itself in a0 register, if the parameter is one and only parameter.
③ Sets the SVC code in d0 register.
④ Issues trap#4 command.
Macro Process Routine
Process macro instruction
① and ② may be omitted if the macro command has no parameters.

◆◆Return Code◆◆

The result of the execution of a macro instruction is stored in the Data Register 0 (d0). When the macro library is used, that value is returned as the return code.

```
long⋮macro ;
long *X ;
     ⋮
if (macro(X)) {......}
     ⋮
```

When the macro library is used, the user program should test the return code. The result of the macro instruction process is returned as the return code.

Although 0 is returned as the return code when a macro instruction is processed normally, some macro instructions reports normal end of the macro processing with a value other than 0.

◆◆ Checking Parameters of Macro Instruction ◆◆

Since a macro instruction is a direct data exchange between a user program and the Compact PMS, an error in parameters may cause malfunctions and/or system down. The Compact PMS makes a parameter check for the specific macros instructions, and aborts the process of the task issuing the associated instruction if an abnormal parameter is found.

4.1.3 Checking parameters

The validity of main parameters of CPMS macro instructions are checked by software. If a parameter is determined as invalid as a result of checking, a macro parameter error is reported and the task which issued the erroneous macro instruction is aborted.

4.2 Types

Type	Name	Explanation
Task control management macros	RLEAS	RLEAS makes the task specified in the parameter enter idle state if the task is in dormant state. Otherwise, RLEAS does not change the stage of the task.
	QUEUE	QUEUE places the task specified in the parameter into a queue if the task in not in dormant state or has not been queued.
	ABORT	ABORT forcibly terminates the task specified in the parameter and makes the task enter dormant state.
Timer management macros	DELAY	DELAY interrupts the task (local task) that issued this instruction, for the time specified in the parameter.
	TIMER	TIMER issues a request to start the task, specified in the parameter, after the specified time elapses, then issues a request to start the task in the specified cycle.
	CTIME	CTIME cancels a TIMER macro instruction.
	STIME	STIME sets or updates the absolute time. *1
	GTIME	GTIME gets the current time.*1
	WAKE	WAKE registers the specified task in the time management table and places it in the scheduled state. *1
	CWAKE	CWAKE cancels the task registered by WAKE.*1
Task state management macros	CHAP	CHAP modifies the priority level of the task specified in the parameter.
	CHMOD	CHMOD modifies the state register level and makes the task enter interrupt not-allowed state.
Factor management macros	SFACT	SFACT sets a factor (0 to 16) to the task specified in the parameter.
	GFACT	GFACT fetches the factor set for the task (local task) that issued this macro instruction.
	USPCHK	USPCHK checks whether the task (local task) that issued this macro instruction is using a stack more than the byte length specified in the parameter.

*1 Optional memory with a clock is required for the 2α and 2αE.

4.2.1 rleas (release)

Function	Checks to determine if the task specified by parameter tn is in dormant (held) state or not. If the task is in dormant status, the status is changed to idle (executable) status. If the task is not in dormant status, no operation is executed.	
Issue Procedure	Assembler	C Language
	⋮ MOVE. L #2, D0 MOVE. L TN, A0 TRAP #4 ⋮	⋮ long tn ; ⋮ rleas (&tn) ; ⋮
Parameters	tn: Double precision integer variable or constant. This is the task number of a called task to be set to the idle status. Specify as follows: 31 8 7 0 0 ←——→ 0 task number	
Return Code	0: Normal end Return code is always zero.	
Parameter Check	If 0 < tn ≤ 128 is not satisfied, the OS assumes a parameter error and aborts the calling task, setting it to the dormant status.	

* Two types of task appear in the descriptions of this macro instruction: calling task and called task. A calling task request the OS (Compact PMS) to change the status of the called task specified by tn in the macro instruction. If tn is not specified the macro instruction parameter processed is the calling task itself. In this case the calling task = called task.

4.2.2 queue (queue (1))

<table>
<tr><td>Function</td><td colspan="2">Changes a called task from the idle status to the execution wait status. The called task (specified by a parameter, tn) is moved to the execution wait queue by the OS unless the called task is in dormant status or is not registered. The called tasks in the execution wait queue are executed in the order of priority levels then queue order (First-In First-Out). Thus if the priority level of the called task is higher than the priority level of the calling task, program control is transferred to the called task. Program control is returned to the calling task, after processing the Queue macro, if the level of the called task is equal to or lower than the level of the calling task. When the called task is placed in the execution wait queue, the "fact" parameter is stored in the initiation factor table. The task initiated by the OS can read the initiation factor by using the GFACT macro instruction.</td></tr>
<tr><td rowspan="2">Issue Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td>⋮
MOVE #3, D0
LEA PARA, A0
TRAP #4
⋮
PARA: DC. L TN
DC. L FACT</td><td>⋮
long tn, fact ;
⋮
queue(&tn, &fact) ;</td></tr>
<tr><td>Parameters</td><td colspan="2">tn : Double precision integer variable or constant; Task number of the called task
fact: Double precision integer variable or constant, Initiation factor to be read by the called task when the task is started</td></tr>
<tr><td>Return Code</td><td colspan="2">0: Normal end (Task is placed in the execution wait queue.)
1: The specified task was in the dormant status.</td></tr>
<tr><td>Parameter Check</td><td colspan="2">If $0 < tn \leq 128$ and $0 \leq fact \leq 16$ are not true, a parameter error is assumed. The calling task is aborted and changed to dormant status.</td></tr>
</table>

(queue (2))

Remarks	. If the called task, specified by the QUEUE macro, has already been queued, the called task will be queued again. A task can be queued once or twice. If a task is queued twice and the first execution of the task is aborted, the task is not executed for the second time. . If a queue instruction is issued with the same factor before the first factor is fetched (by a gfact instruction), fact=0 is used when the task is executed for the second time. The same factor is not stored for the second execution in this case.

4.2.3 abort (abort)

<table>
<tr><td>Function</td><td colspan="2">Forces an end to the execution of the called task, setting the called task to dormant status. If an initiation factor is already stored in the initiation factor table for the called task the factor is deleted.</td></tr>
<tr><td rowspan="2">Issue Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td>⋮
MOVE   #1, D0
MOVE. L TN, A0
TRAP   #4
⋮</td><td>⋮
long tn ;
⋮
abort (&tn) ;
⋮</td></tr>
<tr><td>Parameters</td><td colspan="2">tn: Double precision integer variable or constant. Task number of a called task to be set to dormant status.
Write it as follows:
31 … 8 | 7 … 0
0 ←——→ 0 | task number</td></tr>
<tr><td>Return Code</td><td colspan="2">0: Normal end
Return code is always zero.</td></tr>
<tr><td>Parameter Check</td><td colspan="2">If 0 < tn ≤ 128 is not true, a parameter error is assumed. The issuing task is aborted and changed to dormant status.</td></tr>
<tr><td>Remarks</td><td colspan="2">The status of the called task is changed from the temporarily modified status to the initial status.</td></tr>
</table>

4.2.4 delay (delay)

<table>
<tr><td>Function</td><td colspan="2">Suspends the task issuing this macro instruction for a time specified by a parameter. Control is passed to other task during the suspension, and returned to the calling task after the specified suspension time, if there is no other operable task (a task of a level higher than the issuing task or a task of the same level initiated earlier).</td></tr>
<tr><td>Issue Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td></td><td>⋮
MOVE   #6, D0
MOVE. L T, A0
TRAP   #4
⋮</td><td>⋮
long t ;
⋮
delay (&t) ;
⋮</td></tr>
<tr><td>Parameters</td><td colspan="2">t: Double precision integer variable or constant specifying a period of time for suspension in units of msec.</td></tr>
<tr><td>Return Code</td><td colspan="2">0: Normal end
1: Suspension is not possible because of the timer table (TRB) area is full.
2: Suspension is not possible because the stack save area (ARSB) is full.</td></tr>
<tr><td>Parameter</td><td colspan="2">If $0 < t \leq 86400000$ is not true, a parameter error is assumed. The issuing task is aborted and changed to dormant status.</td></tr>
<tr><td>Remarks</td><td colspan="2">. The parameter must be $0 < t \leq 86400000$ (msec).
. Issuing task not suspended, if the return code is not 0.</td></tr>
</table>

4.2.5 timer (timer)

<table>
<tr><td>Function</td><td colspan="2">Registers the task specified by tn to the timer table (TRB), and initiates it when a specified time elapsed.
The task is initiated each time the cycle time elapsed.
When the cycle time (cyt) is set to 0, the task specified by tn is started only once after elapse of the specified time.
The initiation factor specified by parameter fact is passed to the specified task at initiation.</td></tr>
<tr><td rowspan="2">Issue
Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td>⋮
MOVE   #7, D0
LEA   PARA, A0
TRAP   #4
⋮
PARA: DC.L TN
        DC.L T
        DC.L CYT
        DC.L FACT</td><td>⋮
long tn, t, cyt, fact ;
⋮
timer (&tn, &t,
      &cyt, &fact) ;
⋮</td></tr>
<tr><td>Parameters</td><td colspan="2">tn : Double precision integer variable or constant specifies the task number of the called task.
t   : Double precision integer variable or constant A period of time before the first initiation (msec)
cyt : Double precision integer variable or constant specifies a cycle (msec).
fact: Double precision integer variable or constant specifies the initiation factor to be passed to the called task.</td></tr>
<tr><td>Return
Code</td><td colspan="2">0 : Normal end
1 : Process ended unsuccessfully because the timer table was full.</td></tr>
<tr><td>Parameter
Check</td><td colspan="2">If $0 < tn \leqq 128$, $0 < t \leqq 86400000$, $0 \leqq cyt \leqq 86400000$, $0 \leqq fact \leqq 16$ are not true, or PARA is not on a word boundary, a parameter error is assumed. The issuing task is aborted and changed to dormant status.</td></tr>
<tr><td>Remarks</td><td colspan="2">· The parameter t must be $0 < t \leqq 86400000$ (msec).
· The parameter cyt must be $0 \leqq cyt \leqq 86400000$ (msec).
· The specified task, specified by parameter tn, is not initiated if the specified task is in dormant status.
· A ctime macro instruction is used to cancel a timer macro instruction.</td></tr>
</table>

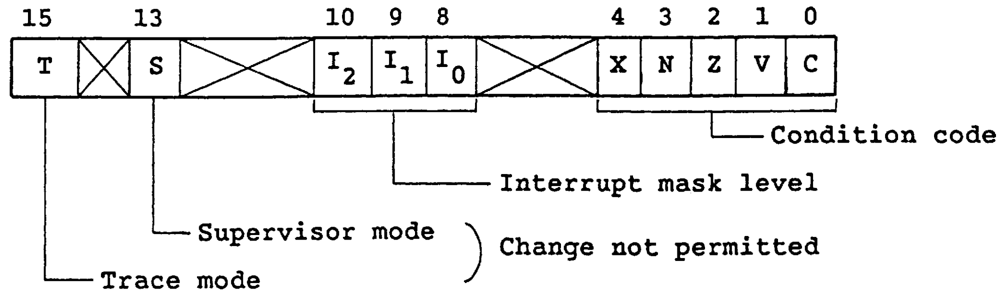
4.2.6 ctime (cancel time)

Function	Cancels the timer macro instruction. The OS checks the timer table (TRB) for values specified by parameters tn and fact. All tasks matching the tn and fact combination of parameters of this macro instruction are deleted from the TRB.	
Issue Procedure	Assembler	C Language
	⋮ MOVE #8, D0 LEA PARA TRAP #4 ⋮ PARA: DC.L TN DC.L FACT ⋮	⋮ long tn, fact ; ⋮ ctime (&tn, &fact) ; ⋮
Parameters	tn : Double precision integer variable or constant. The number of the task, specified in a previous timer macro instruction, to be cancelled. fact: Double precision integer variable or constant. The initiation factor that was specified along with the task number from a previous timer macro instruction.	
Return Code	0: Normal end 1: tn and fact specified by the parameters are not registered in the timer table (TRB).	
Parameter Check	If 0 < tn ≤ 128, 0 ≤ fact ≤ 16 are not true, or PARA is not on a word boundary, a parameter error is assumed. The issuing task is aborted and changed to dormant status.	
Remarks	. This macro instruction cannot abort the task specified by this macro if the task has been initiated and is in execution. The task will not be initiated again after it completes its present execution. . All timer macro instructions matching the tn and fact combination specified in the ctime macro will be cancelled.	

4.2.7 c h a p (change priority level)

Function	Changes the priority level (or execution level) of the task specified by parameter tn. If the task specified by parameter tn is given a higher priority level than the priority level of the task which issued this macro instruction, control is passed to the task which has the higher priority.	
Issue Procedure	Assembler	C Language
	⋮ MOVE #10, D0 LEA PARA, A0 TRAP #4 PARA : DC.L TN DC.L LEVEL ⋮	⋮ long tn, level ; ⋮ chap (&tn, &level) ; ⋮
Parameters	tn : Double precision integer variable or constant task number of the task whose priority level is to be changed. level: Double precision integer variable or constant specifying the priority level being given to the specified task.	
Return Code	0: Normal end 1: Process ended unsuccessfully because the stack save area (ARSB) was full.	
Parameter Check	If 0 < tn ≤ 128, 0 ≤ level ≤ 4 are not true, or PARA is not on a word boundary, a parameter error is assumed. The issuing task is aborted and changed to dormant status.	
Remarks	. If a task issues a chap macro instruction to lower its own priority then control might be passed to another task. . If the task specified by this chap macro instruction has already been initiated, it is assumed that the task was initiated last among the tasks with the same level as the level specified the parameter. . The new priority level specified by this macro is in affect until the specified task ends. The specified task will be executed at its original priority level when it is executed again.	

4.2.8 chmod (change mode (1))

<table>
<tr><td>Function</td><td colspan="2">Changes the contents of the status register of the issuing task. The interrupt mask level can be changed to allow interrupts to be inhibited. Only levels 0 to 3 can be inhibited.</td></tr>
<tr><td rowspan="2">Issue Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td>⋮
MOVE.L #9, D0
MOVE. W WSR, A0
TRAP #4
⋮</td><td>⋮
short wsr ;
⋮
chmod (&wsr) ;
⋮</td></tr>
<tr><td>Parameters</td><td colspan="2">wsr: Single precision integer variable or constant The new contents to be loaded into the status register.</td></tr>
<tr><td>Return Code</td><td colspan="2">Returns the contents of the status register before it is changed.</td></tr>
<tr><td>Parameter Check</td><td colspan="2">None</td></tr>
<tr><td>Remarks</td><td colspan="2">. Allows changes of the interrupt mask level and condition code in the status register (SR).


15
13
10
9
8
4
3
2
1
0
T
S
I2
I1
I0
X
N
Z
V
C
Condition code
Interrupt mask level
Supervisor mode
Trace mode
Change not permitted

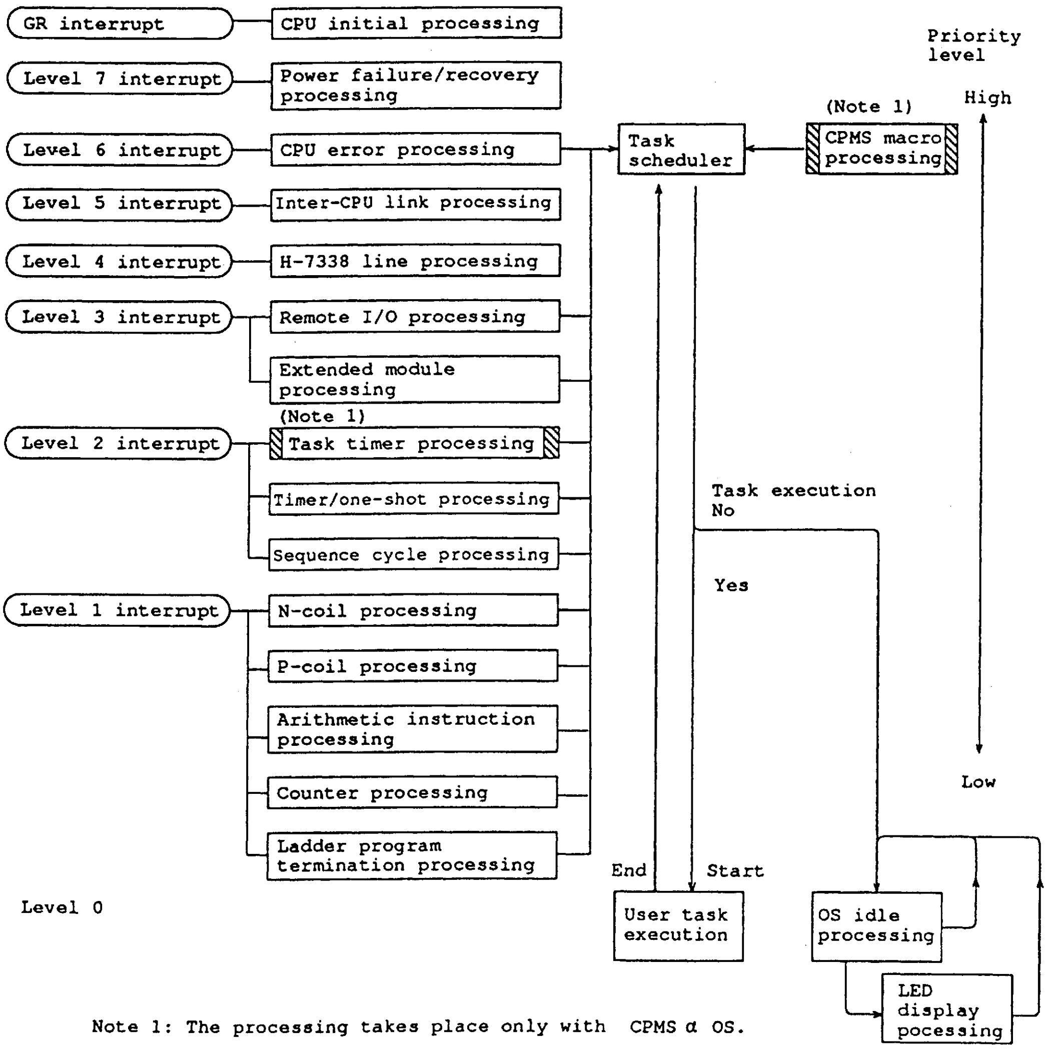
. Only levels 0 up to 3 can be inhibited thus values up to 3 may be loaded. If a level higher than 3 is specified the interrupt mask level is set to 3.
. Although the interrupt mask level may be changed by this macro instruction , the interrupt inhibit time must be kept to a minimum (the interrupt inhibit time must not exceed 2 msec).</td></tr>
</table>

(change mode (2))

Remarks

- The status changed by this macro instruction is effective only within the task issuing the instruction. No other tasks are affected.
- The status changed by this macro instruction is effective until the task issuing this macro instruction is ended or aborted.
- The lower word of the return code contains the contents of the status resister of the issuing task at the time this macro instruction was issued. The upper word contains zero.

The following shows the H-S10/2 α OS organization.



Note 1: The processing takes place only with CPMS α OS.

H-S10/2α OS Organization

4.2.9 sfact (set factor)

<table>
<tr><td>Function</td><td colspan="2">Sets the initiation factor for the specified task specified by parameter tn. The initiation factor is stored in the factor table of the task specified by parameter tn. This initiation factor may be fetched by the GFACT macro instruction.</td></tr>
<tr><td>Issue Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td></td><td><pre>
      ⋮
  MOVE.L #4, D0
  LEA   PARA, A0
  TRAP   #4
      ⋮
  PARA : DC.L TV
         DC.L FACT
      ⋮
</pre></td><td><pre>
      ⋮
  long tn, fact ;
      ⋮
  sfact (&tn, &fact) ;
      ⋮
</pre></td></tr>
<tr><td>Parameters</td><td colspan="2">tn : Double precision integer variable or constant specifies the number of the called task to which the initiation factor is being set.
fact: Double precision integer variable or constant specifies the initiation factor.</td></tr>
<tr><td>Return Code</td><td colspan="2">0: Normal end
1: The specified initiation factor is already registered.
2: The specified called task is in the dormant status.</td></tr>
<tr><td>Parameter Check</td><td colspan="2">If 0 &lt; tn ≤ 128 and 0 ≤ fact ≤ 16 are not true, a parameter error is assumed. The issuing task is aborted and changed to dormant status.</td></tr>
<tr><td>Remarks</td><td colspan="2">. The initiation factor specified can be a number from 0 to 16.
. The initiation factor is not set if the specified task is in dormant status.
. Multiple setting of the same initiation factor is not allowed. If attempted, the return code is set to 1.
. The initiation factor set by this macro instruction is deleted from the initiation factor table when fetched by the gfact macro instruction. All initiation factors are deleted when an abort macro instruction is issued.</td></tr>
</table>

4.2.10 gfact (get factor)

<table>
<tr><td>Function</td><td colspan="2">Fetches an initiation factor that was previously set for the task issuing this gfact macro instruction. The initiation factors are fetched in ascending order of their values, one by one, from the factor table of the task issuing the gfact macro instruction. The fetched factor is deleted from the table. The remaining initiation factors may be fetched by issuing other gfact macro instructions.</td></tr>
<tr><td rowspan="2">Issue Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td>⋮
MOVE.L #5, D0
LEA FACT, A0
TRAP #4
⋮</td><td>⋮
long fact ;
⋮
gfact (&fact) ;
⋮</td></tr>
<tr><td>Parameters</td><td colspan="2">fact: Double precision variable
The factor fetched by this macro instruction is stored in the parameter fact. (From the factor table of the task that issued this macro instruction)</td></tr>
<tr><td>Return Code</td><td colspan="2">0: Normal end
Return code is always zero.</td></tr>
<tr><td>Parameter Check</td><td colspan="2">A parameter error occurs if fact is not on the word boundary. The issuing task aborts and is placed in dormant status.</td></tr>
<tr><td>Remarks</td><td colspan="2">. After a task is initiated, fetch the initiation factors by issuing this macro instruction. End the task after all initiation factors have been fetched.
. If a task issues this macro instruction and no factor has been set for the task, zero is returned to the parameter fact. A return of zero in parameter fact can be used to determine if all initiation factors have been fetched.</td></tr>
</table>

4.2.11 uspchk (user stack pointer check)

<table>
<tr><td>Function</td><td colspan="2">Checks to determine if the task issuing this macro instruction has exceeded the number of bytes of stack area specified by the parameter usebyt. It is the responsibility of the user to determine where to use this macro instruction in a program.</td></tr>
<tr><td>Issue
Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td></td><td>⋮
MOVE.L #11, D0
LEA PARA, A0
TRAP #4
⋮
PARA : DC.L USEBYT
DC.L ADDR
⋮</td><td>⋮
long usebyt, addr ;
⋮
uspchk (&usebyt, & addr) ;
⋮</td></tr>
<tr><td>Parameters</td><td colspan="2">usebyt: Double precision integer variable or constant Specifies the byte length of the stack area reserved for the issuing task.
addr : Double precision variable The number of unused bytes is returned as the execution result of this instruction when it ends normally, and the number of bytes exceeding the limit, when it ends abnormally.</td></tr>
<tr><td>Return
Code</td><td colspan="2">0: Normal end (Space in the stack area) The number of empty bytes is stored in addr.
1: No space in the stack area The number of excessive bytes is returned in addr.</td></tr>
<tr><td>Remarks</td><td colspan="2">. This instruction works most effectively when executed in the deepest location of the program nesting.
. It is advised to delete this instruction from the program upon completion of debugging.
. The user must specify error handling for return code of 1.</td></tr>
</table>

4.2.12 stime (set time (1))

Function	Sets the actual time. The time, specified by a parameter, is set in the optional expansion memory that is equipped with the clock feature.	
Issue Procedure	Assembler	C Language

	⋮
	MOVE.L #13,D0
	LEA PARA, A0
	TRAP #4
	TST.L D0
	⋮
	PARA: DC.L SEC
	DC.W DAY
	DC.W MONTH
	DC.W YEAR
	DC.W WEEK
	⋮
		typedef struct {
		long sec ;
		short day ;
		short month ;
		short year ;
		short week ; } TIME ;
		
		main( )
		{
		long rtn;
		static TIME time {
		SEC,DAY,MONTH,YEAR,WEEK, } ;
		⋮
		rtn = stime(&time) ;
		⋮
		}

Parameters	time ..... Area storing the time to be set. (12 bytes) Specify sec, day, month, year, and week as follows: sec : Time given in units of seconds, assuming 0 a.m. (Midnight) is 0. day : Day month: Month year : Year A.D. week : Day of the week	
Return Code	0: Normal end 1: RTC hardware, not mounted This macro instruction has no effect when the return code is not 0.	

(set time (2))

Parameter Check

A parameter error occurs and the issuing task aborts, going to dormant status, when:

$0 \leq \text{sec} < 86400$ is not satisfied.
$1 \leq \text{day} \leq 31$ is not satisfied.
$1 \leq \text{month} \leq 12$ is not satisfied.
$1900 \leq \text{year} \leq 2199$ is not satisfied.
$1 \leq \text{week} \leq 7$ is not satisfied.

Remarks

(a) The optional expansion memory with clock feature is necessary for using this macro instruction.

(b) Hours and minutes are set in the sec area as follows:
Example: Setting A hour, B minute, C second
SEC = A x 3600 + B x 60 + C

(c) Day of the week is represented by a number as shown below:

Week	1	2	3	4	5	6	7
Day of Week	Sun.	Mon.	Tue.	Wed.	Thu.	Fri.	Sat.

(d) Because of the restrictions on the expansion memory with clock feature, an error might occur in updating the date if the time setting is made as follows:

Status after updating the set date and time	Example
If 23 hr.:59 min.:59 sec., 29th day of any month is set, time is updated to the 1st day of the next month. (Jan. to Dec. except Feb. 29 of a leap year)	Mar. 29 → Apr. 1
If 23 hr.:59 min.:59 sec., 30th day of April, June, September, and November is set, time is updated to the 31st of respective months.	Apr. 30 → Apr. 31
If 23 hr.:59 min.:59 sec. on Feb. 28 is set for a common year, time is updated to Feb. 29.	Feb. 28, '83 → Feb. 29, '83
If 23 hr.:59 min.:59 sec. on Feb. 28 in a leap year is set, time is updated to March 1.	Feb. 28, '84 → Mar. 1, '84

(set time (3))

Remarks

(e) The issue of a stime macro instruction can affect the initiation time of a task that has been scheduled by a wake macro instruction. The effect is dependent upon the relationship between the time setting before the issue of the stime macro instruction and the new time set by the stime macro instruction.

Type of Initiation	Set Time Back	Set Time Forward
Initiation by specified time	(Don't care code specified in wake macro.) If the initiation time becomes greater than 24 hours, because the time was set back, the scheduled initiation is moved to the same time on the day after the new date set. (When the absolute time is specified by the wake macro) The initiation time is not affected by the change of the time setting.	If the initiation time is passed by because the time was set forward, the scheduled initiation is moved to the same time of the next day.
Cyclical initiation by specified time	Same as above.	If the initiation time is passed over because the time was set forward, the scheduled initiation is moved to time when the wake macro first scheduled initiation plus a cycle time that falls after the newly set time.

(f) Change of time must be within the current time +24 hours. If specified otherwise, the affect on scheduled task is the same as if the time change was within 24 hours independent of the date being set ahead or back.

4.2.13 gtime (get time (1))

Function	Fetches the actual time. The time, kept by the expansion memory with clock feature, is stored in an area specified by a parameter.	
Issue Procedure	Assembler	C Language

	⋮
	MOVE.L #14,D0
	LEA PARA, A0
	TRAP #4
	TST.L D0
	⋮
	PARA: DS.L 1 ...... (sec)
	DS.W 1 ...... (day)
	DS.W 1 ...... (month)
	DS.W 1 ...... (year)
	DS.W 1 ...... (week)
	⋮
		typedef struct {
		long sec ;
		short day ;
		short month ;
		short year ;
		short week ; } TIME ;
		main( )
		{
		long rtn;
		static TIME time ;
		⋮
		rtn = gtime(&time);
		⋮
		}

Parameters	time ..... Area for fetching the time (12 bytes). The result of the execution of this instruction is stored in sec, day, month, year, and week as follows: sec : Time is stored in units of seconds, assuming 0 a.m. (Midnight) is 0. day : Day is stored. month: Month is stored. year : Year A.D. is stored. week : Day of the week is stored.	
Return Code	0: Normal end 1: Expansion memory with clock feature is not mounted.	

(get time (2))

<table>
<tr>
<td>Remarks</td>
<td>
(a) The optional expansion memory with clock feature is necessary for using this macro instruction.

(b) Time is stored in the sec area in the following format:

When A hours B minutes C seconds:
sec = A x 3600 + B x 60 + C

(c) Day of the week is represented by a number as shown below:

<table>
<tr><td>Week</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr>
<tr><td>Day of Week</td><td>Sun.</td><td>Mon.</td><td>Tue.</td><td>Wed.</td><td>Thu.</td><td>Fri.</td><td>Sat.</td></tr>
</table>
</td>
</tr>
</table>

4.2.14 wake (wakeup task (1))

Function

Registers the specified task along with mode and factor, all specified by parameters, into the system table (ARB) which changes the task status to scheduled. The specified task is initiated at the time specified by parameter time. If the mode specified is cyclical initiation then the specified task is also initiated at every cycle time after the first initiation. The factor, specified by parameter fact, is passed to the task as the initiation factor whenever it is initiated.

Issue Procedure

Assembler

```
           ⋮
        MOVE.L    #15,D0
        LEA       PARA, A0
        TRAP      #4
        TST.L     D0
           ⋮
  PARA: DC.L      ID
        DC.L      TN
        DC.L      FACT
        DC.L      SEC
        DC.W      DAY
        DC.W      MONTH
        DC.W      YEAR
        DC.W      DUMMY
        DC.L      CYCLE
           ⋮
```

C Language

```
typedef  struct     {
   long     sec     ;
   short    day     ;
   short    month   ;
   short    year    ;
   short    week    ;  }  TIME ;
main(    )
{
long  rtn;
static  long  id, tn, fact, cycle;
static  TIME  time  {
      SEC, DAY, MONTH, YEAR, 0 };
      ⋮
rtn = wake ( &id, &tn, &fact, &time,
           &cycle );
      ⋮
}
```

Parameters

id : Double precision integer variable or constant Initiation mode (0: Time initiation 1: Cyclical initiation)

tn : Double precision integer variable or constant number of the specified task to be initiated

fact : Double precision integer variable or constant Initiation factor to be passed to the specified task being initiated

(wakeup task (2))

Parameters

time : Table consisting of 12 bytes which contains the initiation time

sec : Time is stored in units of seconds, assuming 0 a.m. (Midnight) is 0.
day : Contains day.
month: Contains month.
year : Contains year A.D.
week : No use Set to 0.

cycle: Double precision integer variable or constant Cycle time

(1) Relationship between id, time, and cycle

id	time	cycle	Description
0	Initiation time	0	[Time Wake-Up] A specified task is initiated only once at the time specified by a parameter, time.
1	First initiation time	Time of cyclic initiation after the first initiation time	[Time Cyclic Wake-Up] A specified task is initiated at the time specified by the parameter, time, and afterwards, it is initiated cyclically as specified by cycle.

(2) The initiation time may be set as shown below by using 'Don't Care' code (=-1):

No.	year	month	day	sec	Initiation Time
1	1990	1	10	36610	Initiated at 10 hr.:10 min.: 10 sec. on Jan. 10, 1990.
2	-1 Don't care	1	10	36610	Initiated at 10 hr.: 10 min.: 10 sec. on Jan. 10 in this or next year. (**)
3	(*)	-1 Don't care	10	36610	Initiated at 10 hr.: 10 min.: 10 sec. on 10th day in this or next month.
4	(*)	(*)	-1 Don't care	36610	Initiated at 10 hr.: 10 min. 10 sec. of today or tomorrow. (**)

(*) Any data higher than 'Don't care' code is ignored.

(**) If a specified time precedes the current time, initiation is made in the next year or the next month or tomorrow, otherwise it is made in this year or this month or today.

(wakeup task (3))

Return Code	0: Normal end 1: System table (ARB) is full. 2: Expansion memory with clock feature is not mounted. This instruction is not effective when a return code is not 0.
Parameter Check	The task aborts, going to dormant status, if the parameter conditions are as shown below: Other than $0 \le id \le 1$ Other than $1 \le tn \le 128$ Other than $0 \le fact \le 16$ Other than $0 \le sec < 86400$ Other than $1 \le cycle \le 86400$ Other than $1 \le day \le 31$ Other than $1 \le month \le 12$ Other than $1900 \le year \le 2199$
Remarks	(a) The parameter, fact, must satisfy $0 \le fact \le 16$. (b) The sec within parameter, time, should satisfy $0 \le sec < 86400$. This time must be set in units of seconds starting at 0 a.m. (c) Initiation is not made if the task specified by a parameter, tn, is in dormant status at the scheduled initiation time. (d) To cancel this macro instruction, the cwake macro instruction is necessary. (e) In cyclic initiation, 'cycle' must satisfy $0 < cycle \le 86400$. (f) Even though the tasks registered in the system table (ARB) are set to scheduled status by this macro instruction, the system table is not released unless the registrations are deleted by the cwake macro instruction. (g) If the specified time precedes the current time, the initiation is made at the same time tomorrow.

4.2.15 cwake (cancel wakeup task)

<table>
<tr><td>Function</td><td colspan="2">Cancels a task that was registered by the wake macro instruction. A check is made to determine whether or not the task, specified by parameter tn, is registered in the system table (ARB). If the initiation factor in the table agrees with the factor specified by parameter fact then the task is deleted from the system table and its registration is canceled. All the tasks in the table whose task number and initiation factor agree with those specified by the parameters are deleted.</td></tr>
<tr><td rowspan="2">Issue
Procedure</td><td>Assembler</td><td>C Language</td></tr>
<tr><td>⋮
MOVE.L #16,D0
LEA PARA, A0
TRAP #4
TST.L D0
⋮
PARA: DC.L TN
DC.L FACT
⋮</td><td>main( )
{
long tn, fact, rtn;
⋮
rtn = cwake (&tn, &fact);
⋮
}</td></tr>
<tr><td>Parameters</td><td colspan="2">tn : Double precision integer variable or constant specifies a task number.
fact: Double precision integer variable or constant specifies the initiation factor.</td></tr>
<tr><td>Return
Code</td><td colspan="2">0: Normal end
1: Unsuccessful end. No tasks were found in the system table (ARB) matching the specified task number and initiation factor.
2: Expansion memory with clock feature is not mounted.</td></tr>
<tr><td>Parameter
Check</td><td colspan="2">The issuing task aborts going to the dormant status if the parameters are:
1 Other than 1 ≤ tn ≤ 128
2 Other than 0 ≤ fact ≤ 16</td></tr>
<tr><td>Remarks</td><td colspan="2">(a) It is not possible, with this macro instruction, to abort a scheduled task that is already initiated and in execution. The following scheduled initiations are cancelled.
(b) A number of wake instructions may be cancelled by one cwake instruction, if their tn and fact agree with those specified by this macro instruction.</td></tr>
</table>

4.2.16 rserv (reserve (1))

Function	Reserves an area of memory for the task issuing this macro instruction. If the issuing task has already reserved the resource, by using the rserv macro instruction, this instruction is ignored. If the resource is not yet reserved a check is made to determine if the intended area is reserved by another task, such that the area specified by the parameters is reserved only when the area is not reserved by any other task. If the area is reserved by another task the task issuing this macro instruction is set to wait status until the specified area is released by a free macro instruction. The issuing task, which is in wait status, reserves the specified area and receives control when the area is released and all the area specified is available for its reservation. Even though an area is reserved by this instruction, access to the area by other tasks cannot be prevented since the reservation of the area by this macro instruction is different from the interlock by the hardware system. To prevent the area from being accessed, a rule "to reserve an area by using this macro instruction before making an access to this area" must be observed by all tasks.

(reserve (2))

Issue Procedure

Assembler

```
               ⋮
        MOVE.L    #17,D0
        LEA       PARA, A0
        TRAP      #4
        TST.L     D0
               ⋮
PARA:   DC.L      CASE
        DC.L      TYPE1  ⎫
        DC.L      TABLE1 ⎬ Case #1
        DC.L      TOP1   ⎪
        DC.L      LAST1  ⎭
        DC.L      TYPE2  ⎫
        DC.L      TABLE2 ⎬ Case #2
        DC.L      TOP2   ⎪
        DC.L      LAST2  ⎭
               ⋮
```

C Language

```
                 ⋮
typedef struct          {
      long   type  ;
      long   *table ;
      long   top   ;
      long   last  ; } para ;


main(    )
   {
   long  case, rtn;
   static  para  para1       {
         0, &table1(0), top1, last1};
   static  para  para2       {
         0, &table2(0), top2, last2};
                ⋮
   rtn = rserv (&case, &para1,
                  &para2, ...);
                ⋮
   }
```

Parameters

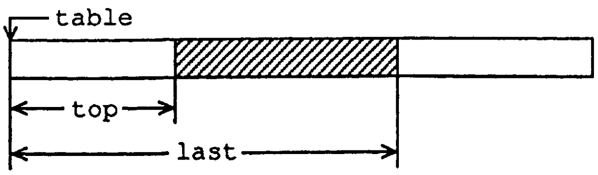
case : Double precision integer variable or constant
The number of cases of the resource to be occupied

type : Double precision integer variable or constant
Type of the resource to be occupied
'0' must be specified in CPMS.

table: Double precision integer variable or constant
Table head address of the area to be occupied

top : Double precision integer variable or constant
Relative byte address from the head of the area to be occupied

last : Double precision integer variable or constant
Relative byte address from the head of the area to be occupied
The shaded portion in the following figure is occupied.



(reserve (3))

Return Code	0: Reserve succeeded 1: The rserv macro has been issued. 2: A parameter contains an illegal specification. 3: ARB is full. This macro instruction has no effect if the return code is not 0.
Parameter Check	The issuing task aborts the execution, going to dormant status if the parameter does not satisfy 1 ≤ case ≤ 32.
Remarks	(a) This macro instruction loses its effect when the task ends (Exit or Abort). (b) If the intended area overlaps the area occupied by other task, the area is not occupied. The task which issues this macro instruction is set to wait status (Suspension of Execution) (c) The same task cannot issue this instruction twice in succession. The resource to be reserved should be reserved by one rserv instruction. However, it is permitted to issue this instruction again after all the resources are released by the free macro instruction. (d) It is advised to make the term of the reservation by this instruction as short as possible. (e) This instruction does not control the OS and hardware. (f) Avoid issuing this macro instruction after issuing a macro instruction which suspends the execution of other tasks. It may cause a deadlock if the resource is already reserved by the associated task. (g) If exited without releasing all the reserved resources by using the free macro instruction, "RSV ERR" is shown on the console LED of CPU.

4.2.17 free (free (1))

Function	Cancels the reservation by the rserv macro instruction and releases the reserved resources for use by other tasks. The reserved resources are released if the parameters of this instruction indicate resources reserved by a rserv macro instruction previously issued from the same task.

Issue Procedure

Assembler

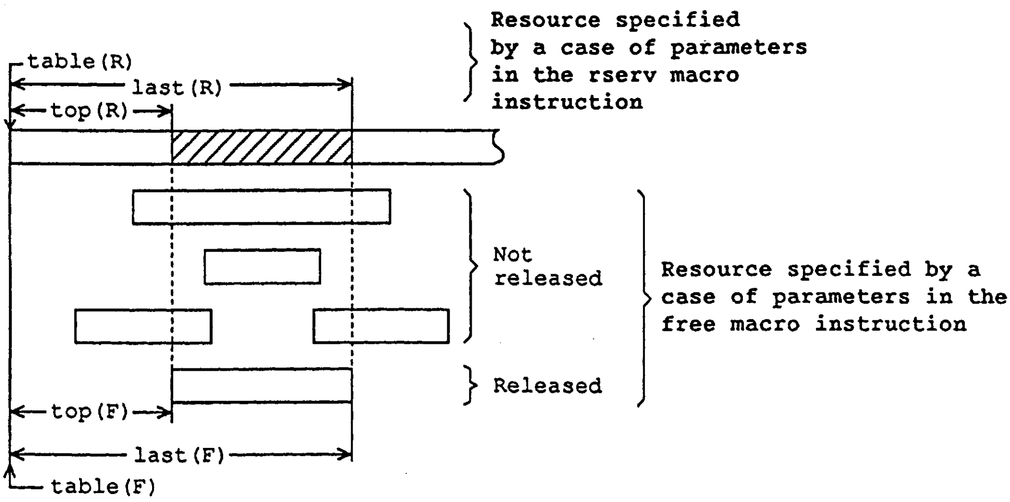
```
              ⋮
        MOVE.L    #18,D0
        LEA       PARA, A0
        TRAP      #4
        TST.L     D0
              ⋮
PARA:   DC.L      CASE
        DC.L      TYPE1  )
        DC.L      TABLE1 |
        DC.L      TOP1   } Case #1
        DC.L      LAST1  )
        DC.L      TYPE2  )
        DC.L      TABLE2 |
        DC.L      TOP2   } Case #2
        DC.L      LAST2  )
              ⋮
```

C Language

```
          ⋮
typedef struct          {
        long   type   ;
        long   *table ;
        long   top    ;
        long   last   ; } para ;
          ⋮
main(     )
{
long  case, rtn;
static  para  para1     {
          0, &table1(0), top1, last1};
static  para  para2     {
          0, &table2(0), top2, last2};
          ⋮
rtn = free (&case, &para1,
               &para2, ...) ;
          ⋮
```

Parameters	The same as the rserv macro instruction
Return Code	0: Cancel of reservation succeeded for all cases specified. 1: At least one of the specified parameters does not indicate a resource reserved by this task. Reservation of the resource which satisfies the specified parameter is cancelled, even when return code = 1.

(free (2))

Parameter Check	The calling task aborts, going to dormant status, if 1 ≤ case ≤ 32 is not satisfied.
Remarks	(a) When the reserved resources are released by this macro instruction, the task waiting to reserve those resources are released from the wait status. (b) The parameters of the free macro instruction must agree with those specified in the rserv macro instruction. If not, the resources in reservation can not be released.  table (R) last (R) top (R) Resource specified by a case of parameters in the rserv macro instruction Not released Released Resource specified by a case of parameters in the free macro instruction top (F) last (F) table (F) Conditions of Releasing a Resource table(R) = table(F) top(R) = top(F) last(R) = last(F)

4.2.18 mvmem (move memory (1))

Function	Writes data into an area of memory as specified by the parameters for this macro instruction. This instruction enables data to be written into protected memory (except OS program area).	
Issue Procedure	Assembler	C Language

	⋮
	MOVE.L #19,D0
	LEA PARA, A0
	TRAP #4
	TST.L D0
	⋮
	PARA: DC.L WNO
	DC.L DADDR
	DC.L SADDR
	⋮
		⋮
		main( )
		{
		long rtn, wno, saddr, daddr;
		⋮
		rtn = mvmem (&wno, &daddr, &saddr);
		⋮
		}

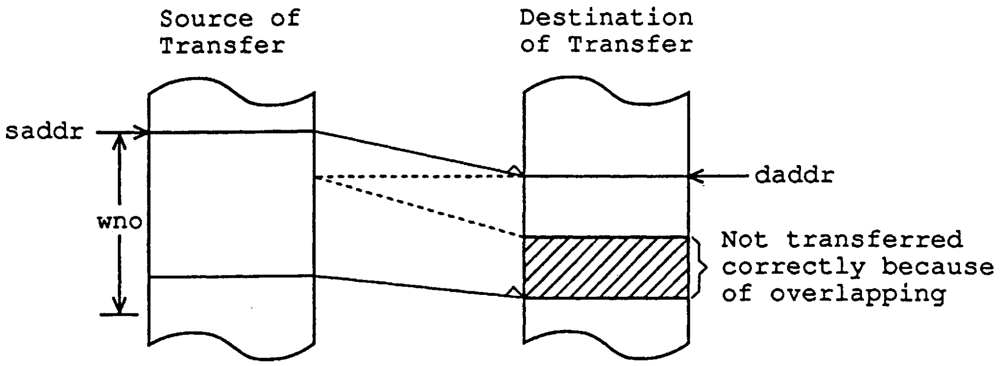
Parameters	wno : Double precision integer variable or constant Specifies the number of words to be transferred in units of words (16 bits). daddr: Double precision integer variable or constant Specifies the leading address of the data destination saddr: Double precision integer variable or constant Specifies the leading address of the data source	
Return Code	0: Normal end Always 0	
Parameter Check	The issuing task aborts its execution and goes to dormant status if: . 0 < wno ≤ 256 is not satisfied, the area specified by daddr is write-protected by the OS.	

(move memory (2))

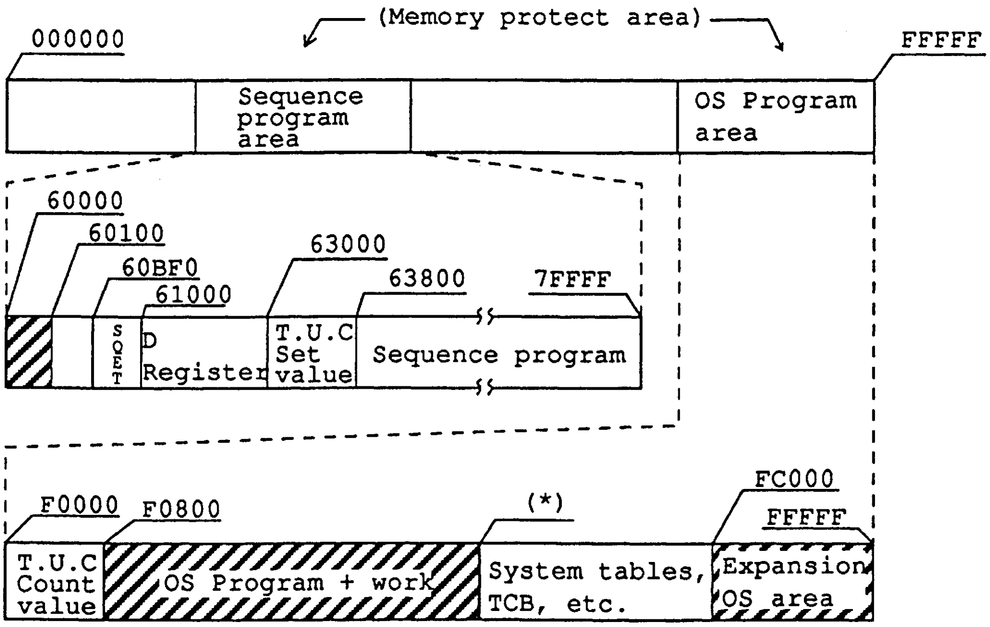
Remarks

(a) The number of words to be transferred must satisfy 0 < wno ≤ 256.

(b) The source and destination areas can not be overlapped. If there is any overlap, data might not be written correctly, as the transfer of data always begins with the lower address.



(c) This instruction enables data to be written into the memory protect area regardless of whether the protect switch is in ON or OFF state. The OS program area and the system area re always write-protected.



▨ : Write-prohibited area

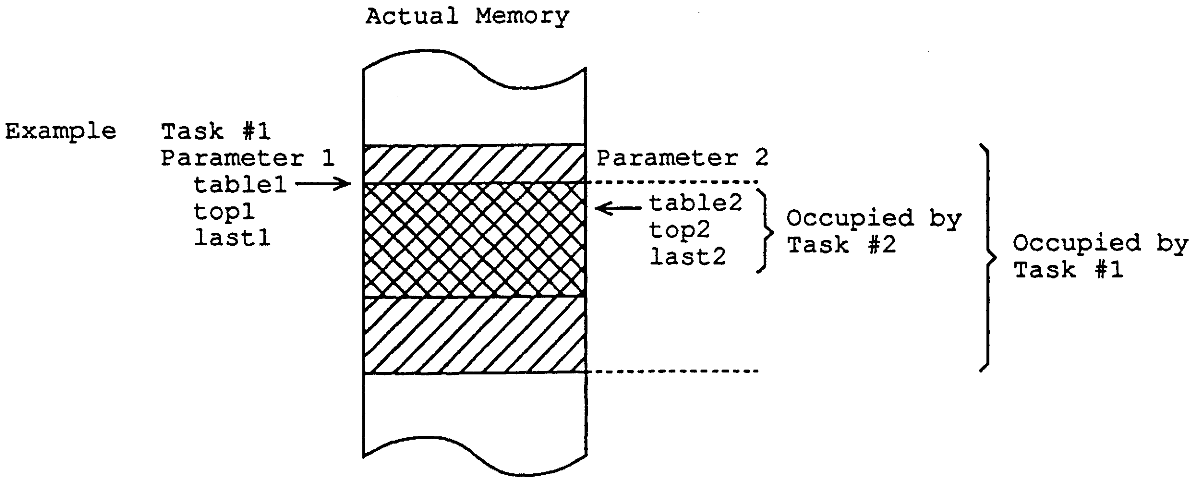
(*) : Differs according to OS versions.

4.3 Supplement

4.3.1 Relationship between rserv and free

- RSVB (Reserve Block) stores the parameters from each issuance of rserv macro. It is sized for a maximum of 32 cases of reserved resources. An error occurs if an attempt is made to reserve more than 32 areas without issuing a free macro instruction.

- If the value of the parameter table of the rserv macro instruction are different from that of the free macro instruction, they are treated as different areas. Thus if the data table is not loaded correctly abnormal processing would be expected.



In the above example, it is clear that the areas specified by the parameters 1 and 2 are actually overlapped. However, OS treats them as different areas because the values of parameters 1 and 2 are not equal, and both Task #1 and Task #2 reserve their own areas. This makes the normal exclusive control impossible.

- If a task with the resource reserved by the rserv instruction ends without cancelling the reservation of the resource by using the free macro instruction, "RSV ERR" is displayed on the Console LED of the CPU.

4.3.2 Relationship between wake and cwake

- A maximum of 8 tasks can be queued for time initiation by the clock feature (scheduled status). Caution should be exercised because a system table (ARB: Alarm Recording Block) error occurs if use of more than 8 tasks is attempted.

- Once a task is scheduled by the wake macro instruction, it is removed from the schedule only by the issue of the cwake macro instruction or when the CPU is reset (power outage/recovery). Therefore a task is always queued by the OS at its scheduled time unless it has been aborted by an abort macro instruction. (It is not actually initiated because it is not in the idle status.)

- Caution should be exercised because the task scheduling by the wake macro instruction is affected by the stime macro instruction. When issuing the stime macro instruction, it is best to issue it in the initial task or before issuing the Wake macro instruction to a task.

- A task is initiated next day if non-existent day is set, since a date check is made within the range of 1-31.

  Example: If "10:00 April 31st 1988" is specified, the task is initiated at 10:00 on May 1. Caution should be used because the task is not initiated, if specified April 31 using Don't care code.

4.3.3 Relationship between stime and gtime

- The relation between a date and day of the week is not checked within the OS. Users must use caution when setting with the stime macro instruction.

- The time is set to 0 hr:0 min.:0 sec., Jan. 1, 1900 when the system is delivered to a user. The user is responsible for setting it correctly using the stime macro instruction.

- A leap year is taken into consideration only when the days in a year can be divided by 4 without a remainder, since the H-S10/2α.2αE is not provided with the calendar feature.

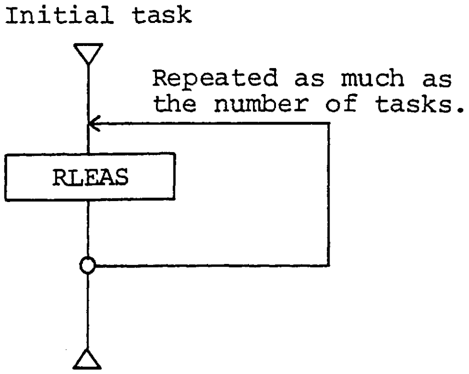
4.4 How to Use Macro Instructions

■ RLEAS (release)

The RLEAS macro instruction changes the state of a task from dormant to idle. All tasks other than the initial task (task number 1) are in dormant state (task execution inhibited) when the CPU power supply is turned on. As a result, CPU cannot perform processing.

Generally, the initial task (started automatically by CPMS when the CPU power supply is turned on) makes all other tasks constructing the system enter idle state (task execution allowed).

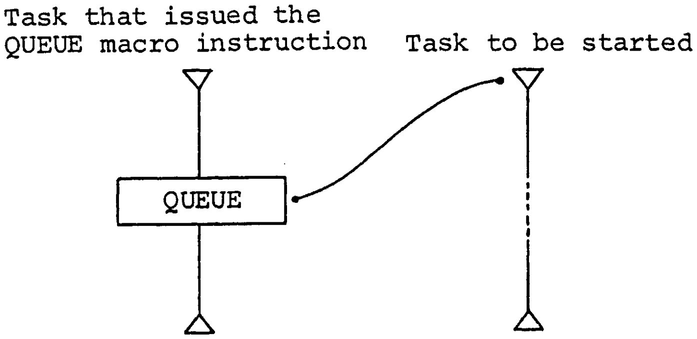
[Example]



■ QUEUE (queue)

The QUEUE macro instruction starts a task that is in idle state (task execution allowed).

[Example]

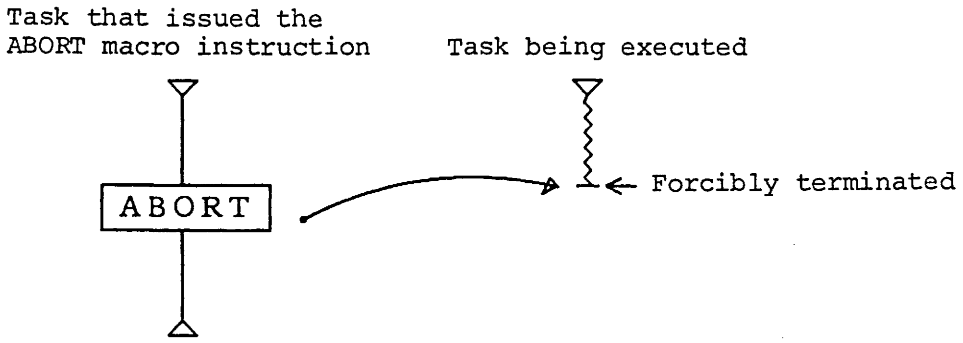


- ● If the task to be started is in dormant state (task execution inhibited), the task is not started.

■ ABORT (abort)

The ABORT macro instruction makes a task enter dormant state (task execution inhibited). If the task to be aborted by the macro instruction is being executed, the task is forcibly terminated and made to enter dormant state.

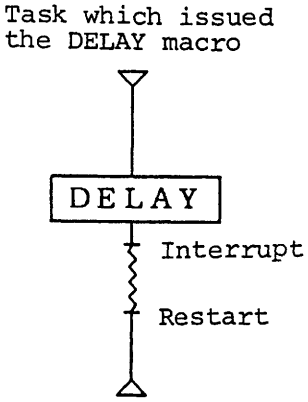
[Example]



■ DELAY (delay)

The DELAY macro instruction interrupts the task, which issued the DELAY macro instruction, and restarts it after the specified time.

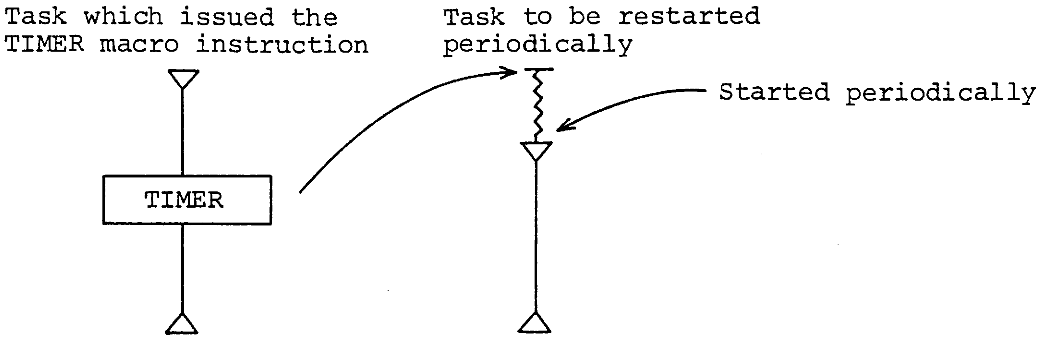
[Example]



■ TIMER (timer)

The TIMER macro instruction starts the specified task when the specified time elapses, then repeats starting the task in the specified cycle. In other words, this macro instruction starts a task periodically.

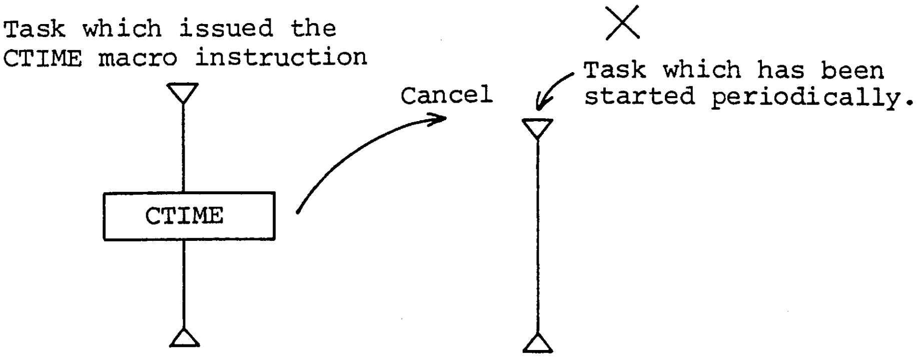
[Example]


Task which issued the TIMER macro instruction
Task to be restarted periodically
Started periodically
TIMER

■ CTIME (cancel time)

The CTIME macro instruction cancels a request which was issued by a TIMER macro for periodically starting a task. After the CTIME macro instruction is issued, the task which has been periodically started is no longer started.

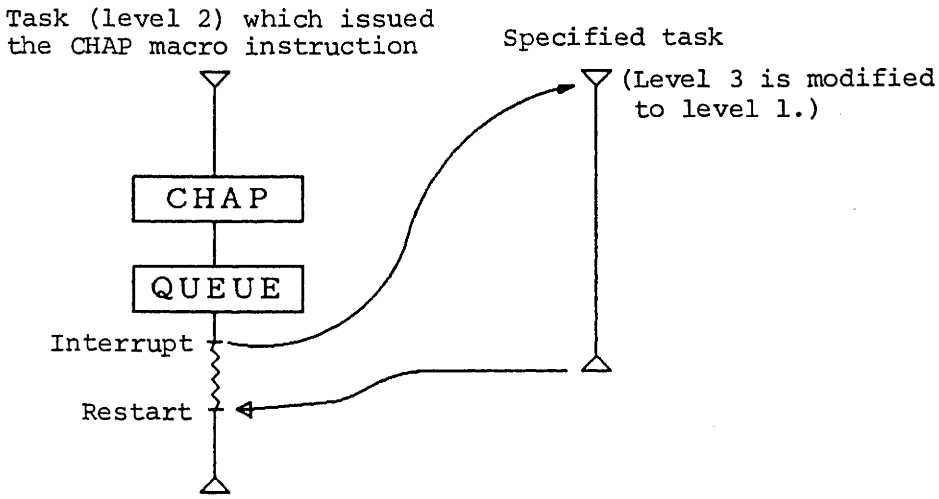
[Example]


Task which issued the CTIME macro instruction
Cancel
Task which has been started periodically.
CTIME

■ CHAP (cnange priority level)

The CHAP macro instruction temporarily modifies the priority level of the specified task. The priority level of a task is used to determine the execution order of the task. If an attempt is made to start two tasks at the same time, the task which has the higher priority level is executed first.

[Example]


Task (level 2) which issued the CHAP macro instruction
Specified task
(Level 3 is modified to level 1.)
CHAP
QUEUE
Interrupt
Restart

In the above example, the level of the specified task is raised higher than the level of the local task and the specified task is started by a QUEUE macro instruction, so that the specified task is executed while the local task is made to wait.

■ CHMOD (change mode)

The CHMOD macro instruction modifies the contents of state register. The state register stores a conditions code (indicates overflow, zero, or negative) or an interrupt mask level (there are 8 levels). By rewriting the interrupt task, interrupt with the specified mask level or less can be inhibited. This operation is useful to execute a job prior to hardware.

Hardware interrupts are in the following levels:

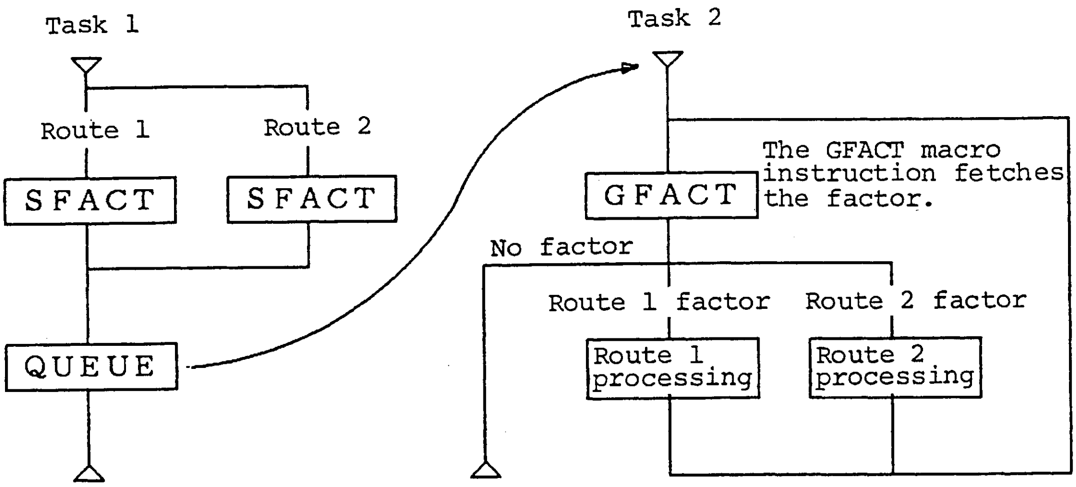
Low priority	Level 1 :	P coil
	Level 1 :	S mode termination
	Level 2 :	Timer
High priority	Level 3 :	Remote I/O

■ SFACT (set factor)

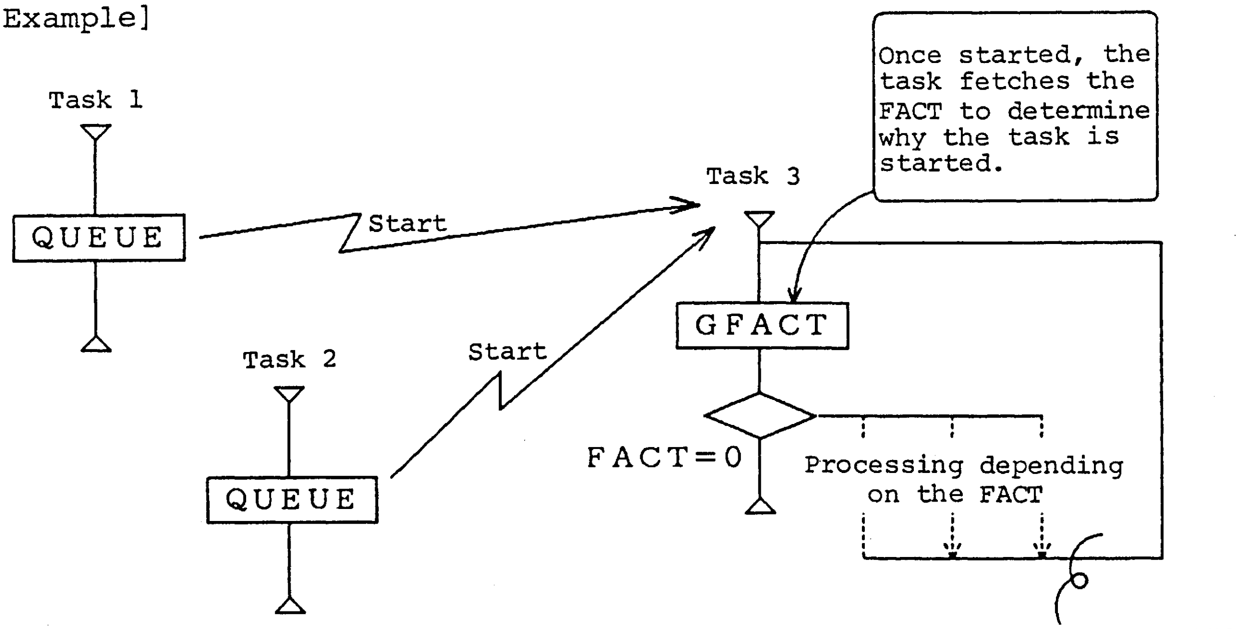
■ GFACT (get factor)

The SFACT macro instruction sets the start cause (factor) of the specified task. The GFACT macro instruction fetches the set factor. A factor is used to determine why a task is started and to choose processing allocated to the factor.

[Example]



[Example]


Task 1
QUEUE
Start
Task 2
Start
QUEUE
Task 3
Once started, the task fetches the FACT to determine why the task is started.
GFACT
FACT=0
Processing depending on the FACT

Task 3 uses a GFACT macro instruction to determine the task which started task 3 (task 1 or task 2). In other words, if the FACT used when task 1 starts task 3 is different from the FACT used when task 2 starts task 3, task 3 checks the FACT whether task 1 or task 2 has started task 3.

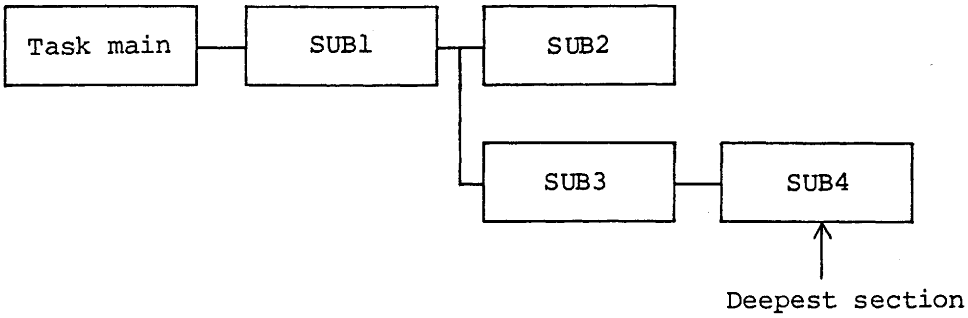
The task must check the FACT again before it terminates.

■ USPCHK (user stack pointer check)

The USPCHK macro instruction checks whether the local task is using the stack area more than the size specified by the local task. When a task uses an USPCHK macro instruction at the task's deepest section, the stack area size required by the task can be estimated.

[Example]

Assume that a task has the following program configuration:



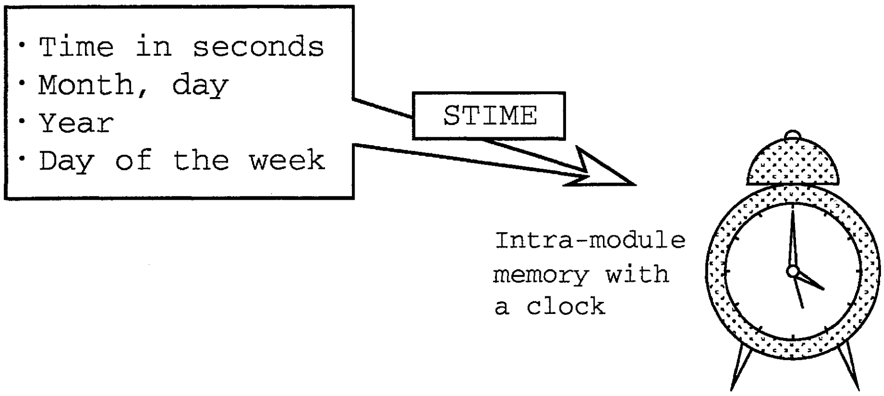
In the above example, SUB4 is the deepest section of the task. The stack use state can be obtained by issuing an USPCHK macro instruction in SUB4. However, if a lot of local area is used by another route, an USPCHK macro instruction must be used in the deepest section of that route.

● Local area: Program work area reserved in the stack area

■ STIME (set time) (Execution of this macro instruction requires memory with a clock.)

The STIME macro instruction sets the real time specified by parameters in memory with a clock that manages the time of day. That is, this macro instruction sets or updates the absolute time.

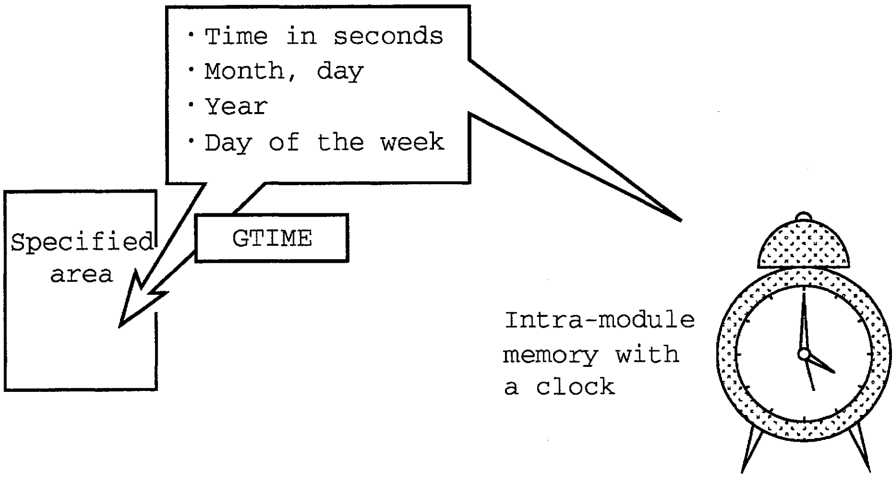
The following parameters are available:



■ GTIME (get time) (Execution of this macro instruction requires memory with a clock.)

The GTIME macro instruction stores the time of day, managed in memory with a clock, in the area specified by a parameter. That is, this macro instruction fetches the real time.

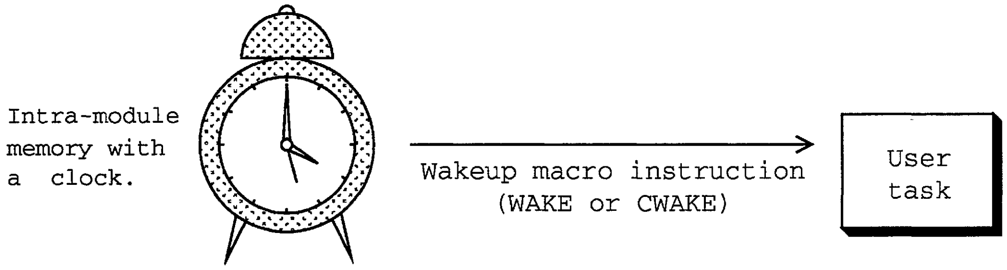
After execution, the following values are stored:



WAKE (wakeup task) (Execution of this macro instruction requires memory with a clock.)

The WAKE macro instruction registers a task having the task number specified by a parameter in a system table, then places the task in the scheduled state. The task in the scheduled state will be started at the time of day specified by parameters.

When cyclic start is specified, the task is started each time the specified cycle time elapses.

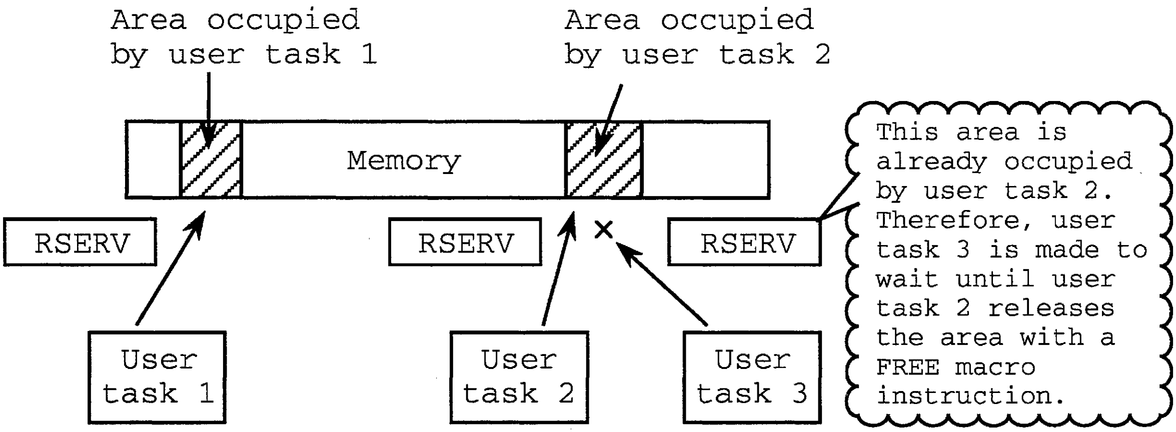

Intra-module memory with a clock.
Wakeup macro instruction (WAKE or CWAKE)
User task

CWAKE (cancel wakeup task) (Execution of this macro instruction requires memory with a clock.)

The CWAKE macro instruction deletes all tasks identified by the task number, and also deletes start factor parameters from the table in which they were registered by a WAKE macro instruction. That is, this macro instruction cancels a WAKE macro instruction.

■ RSERV (reserve)

In task-shared memory, the RSERV macro instruction reserves the area specified by a parameter.

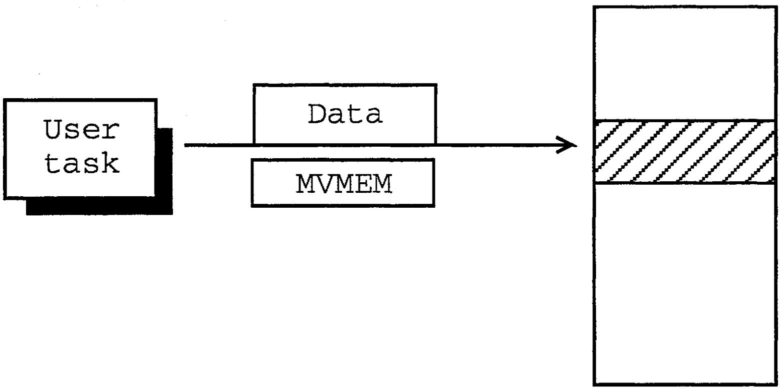

Area occupied by user task 1
Area occupied by user task 2
Memory
RSERV
RSERV
×
RSERV
User task 1
User task 2
User task 3
This area is already occupied by user task 2. Therefore, user task 3 is made to wait until user task 2 releases the area with a FREE macro instruction.

■ FREE (free)

The FREE macro instruction releases memory reserved by an RSERV macro instruction. That is, this macro instruction cancels the effect of an RSERV macro instruction.

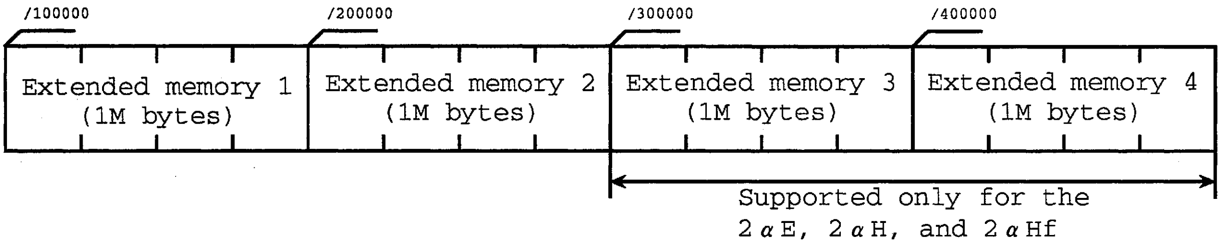
■ MVMEM (move memory)

The MVMEM macro instruction copies the data specified by a parameter to the specified area. This macro instruction enables data to be written to protected memory (excluding the program area for the operating system).

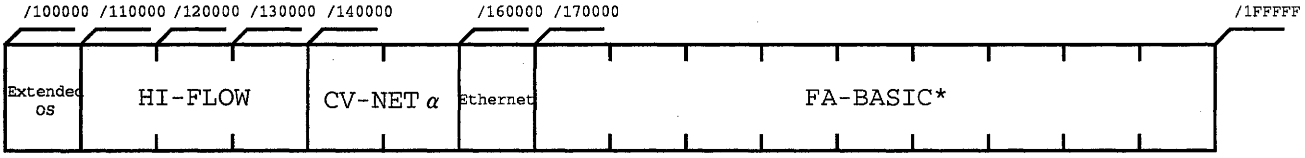


5 APPENDIX

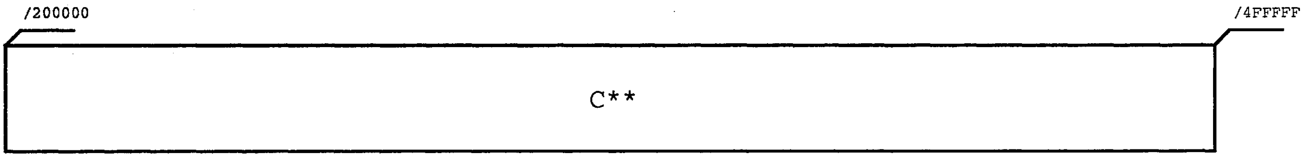
APPENDIX 1 EXTENDED MEMORY ALLOCATION IN THE H-S10/2α SERIES


/100000
/200000
/300000
/400000
Extended memory 1 (1M bytes)
Extended memory 2 (1M bytes)
Extended memory 3 (1M bytes)
Extended memory 4 (1M bytes)
Supported only for the 2αE, 2αH, and 2αHf

Extended memory 1



Extended memories 2 to 4



* The user can allocate the FA-BASIC area within the range /110000 to /1FFFFF.

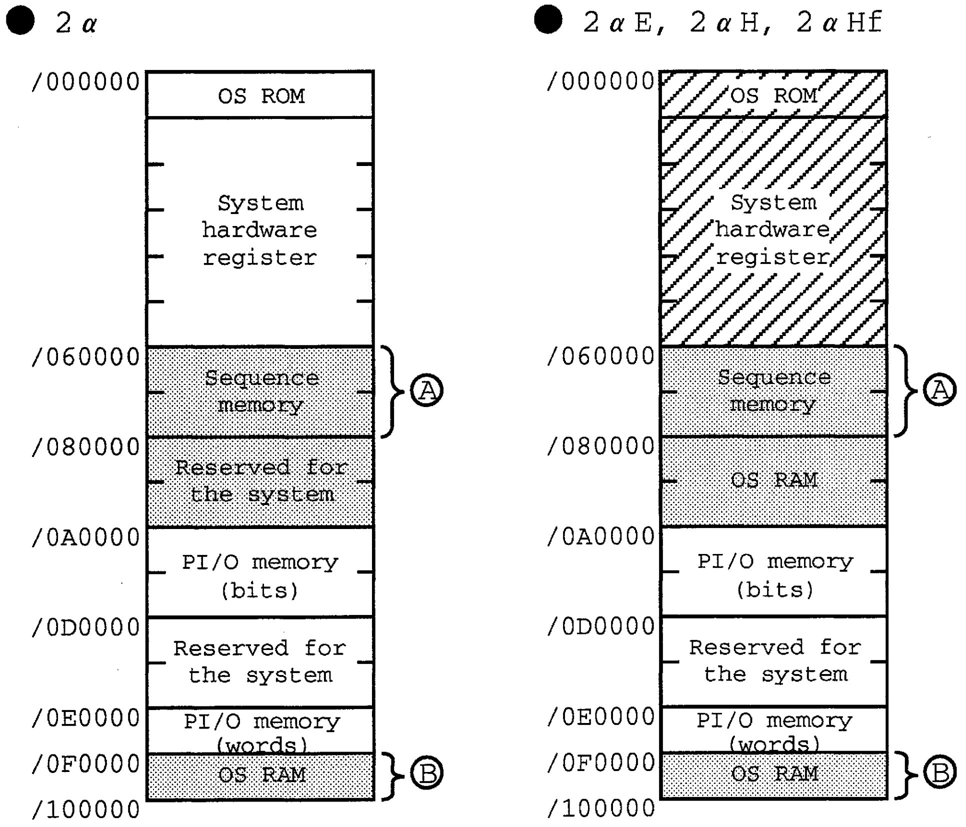
** The user can allocate the C area within the range /110000 to /4FFFFF.

APPENDIX 2 PROTECT KEYSWITCH

(1) Purpose

The protect keyswitch protects the system area from being destroyed inadvertently by user tasks in C or FA-BASIC. The protect keyswitch cannot be used in arithmetic functions (including user arithmetic functions).

(2) Effective range of memory protection



● 2α

● 2αE, 2αH, 2αHf

▨ : Memory is protected for a read and write.

░ : Memory is protected for a write, but not for a read.

Ⓐ : The LPET, DW register, and TUC setting are included.

Ⓑ : The UFET, PRET, and TUC values are included.

(3) Protect errors (indicated by the PROT ERR indicator)

When the protect keyswitch is turned on but a user task attempts to access a protected area, a protect error occurs. In this case, only the user task is aborted.

When the user task is registered in the user arithmetic function table (UFET) rather than the program edition table (PRET), however, protection is disabled.

APPENDIX 3 DHP INFORMATION

No.	Function	DHP code (D7.L)				Remarks
		No.	Parameter1	Parameter2	Parameter3	
1	QUEUE	01	TN	-	-	TN: Task number
2	Task exit	02	ETN	-	-	ETN: End task number
3	ABORT	03	TN	-	-	
4	RLEAS	04	TN	-	-	
5	SFACT	05	TN	-	-	
6	GFACT	06	CTNO	-	-	CTNO: Number of the macro-issuing task
7	DELAY	07	CTNO	-	-	
8	TIMER	08	TN	-	-	
9	CTIME	09	TN	-	-	
10	CHMOD	0A	CTNO	-	-	
11	CHAP	0B	LEVEL	-	-	LEVEL: New level
12	Initial task start	0C	TN	-	-	
13	Task restart	0D	TN	-	-	
14	CPMS initialization	0E	00	-	-	
15	Idle	0F	00	-	-	
16	BCAN	10	TN	-	-	
17	USPCHK	11	CTNO	-	-	
18	STIME	12	CTNO	-	-	
19	GTIME	13	CTNO	-	-	
20	WAKE	14	TN	-	-	
21	CWAKE	15	TN	-	-	
22	RSERV	16	CTNO	-	-	
23	FREE	17	CTNO	-	-	
24	WATE	18	CTNO	-	-	
25	POST	19	CTNO	-	-	
26	DEFCD	1A	CTNO	-	-	
27	ENQ	1B	CTNO	-	-	
28	DEQ	1C	CTNO	-	-	
29	SUSP	1D	TN	-	-	
30	RSUM	1E	TN	-	-	
31	Parameter error	1F	CTNO	EC		EC: Macro parameter error code
32	Error interrupt	20	TN	ISW		IN = 0 indicates a system failure ISW: Interrupt factor
33	Extended board interrupt	21				This function is incorporated in the operating system that supports the extended board.
34	MVMEN	00	CTNO	-	-	
35	Coil interrupt	22	COIL	CNO		COIL=0:P coil interrupt COIL=1:N coil interrupt COIL=2:Counter interrupt COIL=3:Operation instruction interrupt CNO=Coil number
36	Timer interrupt	23	K	-	-	K = 1: SEQ timer K = 2: T.U timer K = 3: Task timer interrupt
37	RI/O termination	24	00	-	-	
38	SEND interrupt	25	CSPN	-	-	CSPN: Number of terminated S-mode program

Reserved for future expansion

Applied when DHPMD = 1

Applied when DHPMD = 2

APPENDIX 4 MACRO INSTRUCTIONS AND THEIR EXECUTION TIMES

The execution times listed below were determined from processing durations (numbers of steps) under the processing conditions shown below that were imposed on the execution of individual macro instructions.

No.	Condition	2αH 2αHf (μs)	2αE(μs)	2α (μs)	Remarks (without DHP, with parameter check)
1	From the time a QUEUE macro instruction is issued until the task starts.	115.6	154.9	442.9	RLEAS was already issued to the target task. The target task has a higher priority than the macro-issuing task.
2	From the time a macro instruction terminates until the task enters the idle state	32.5	43.5	124.4	The task terminates normally and is not started more than once.
3	From the time an ABORT macro instruction is issued until control returns to the macro-issuing task	68.6	91.9	262.8	RLEAS was already issued to the target task.
4	From the time an SFACT macro instruction is issued until control returns to the macro-issuing task	64.8	86.8	248.2	Only the macro-issuing task is present.
5	From the time a GFACT macro instruction is issued until control returns to the macro-issuing task	61.4	82.3	235.3	Applicable when fact = 8 is fetched
6	From the time a DELAY macro instruction is issued until the task enters an idle state.	94.1	126.1	360.5	Only the macro-issuing task is present.
7	From the time a timer interrupt is generated until the task restarts.	124.2	166.4	475.8	The task being delayed is restarted. The task is idle when an interrupt is generated.
8	From the time a TIMER macro instruction is issued until control returns to the macro-issuing task	107.8	144.4	412.9	Only the macro-issuing task is present.
9	From the time a timer interrupt is generated until the task starts the first time.	119.3	159.8	456.9	The task is idle when an interrupt is generated. Only one task is scheduled. RLEAS was already issued to the task.
10	From the time a CTIME macro instruction is issued until control returns to the macro-issuing task	64.6	86.5	247.3	Only one task is scheduled currently.
11	From the time a CHMOD macro instruction is issued until control returns to the macro-issuing task	52.4	70.2	200.7	Only the macro-issuing task is present.
12	From the time a CHAP macro instruction is issued until control returns to the macro-issuing task	110.2	147.6	422.0	There are no other tasks. RLEAS was already issued to the target task.
13	From the time an RLEAS macro instruction is issued until control returns to the macro-issuing task	64.3	86.1	246.2	The target task is in a dormant state.
14	From the time a USPCHK macro instruction is issued until control returns to the macro-issuing task	62.3	83.5	238.7	Only the macro-issuing task is present.
15	From the time an STIME macro instruction is issued until control returns to the macro-issuing task	137.7	184.5	527.5	The task is not scheduled again.
16	From the time a GTIME macro instruction is issued until control returns to the macro-issuing task	54.8	73.4	209.9	Only the macro-issuing task is present.
17	From the time a WAKE macro instruction is issued until control returns to the macro-issuing task	101.3	135.7	388.0	Only the macro-issuing task is present. The ARB is empty (no task is waiting).
18	From the time a CWAKE macro instruction is issued until control returns to the macro-issuing task	67.6	90.5	258.8	Only one task is scheduled currently.
19	From the time an RSERV macro instruction is issued until control returns to the macro-issuing task	77.2	103.4	295.6	The parameter applies to one case only. RSERV can be issued. There are no other tasks waiting for resources to be released.
20	From the time a FREE macro instruction is issued until control returns to the macro-issuing task.	96.7	129.6	370.5	Only one case is occupied. There are no tasks waiting for resources to be released.
21	From the time an MVMEN macro instruction is issued until control returns to the macro-issuing task	68.6	91.9	262.8	WNO=1
22	From the time an interrupt to start the task at the specified time of day is issued until the task first starts.	137.8	184.6	527.8	The task is started at the specified time of day, but not cyclically. Other tasks are not started.
23	From the time GR is issued until the task is first started.	226636.4	303607.2	868060.0	RTC(1M) was already issued. There is no parity error. Ladder circuits are not used.