APOLLO GUIDANCE COMPUTER

Information Series

ISSUE 12A

JOB CONTROL AND TASK CONTROL

FR-2-112A

17 April 1964

CONTENTS

ILLUSTRATIONS

TABLES

ATTACHMENTS

12-1. INTRODUCTION

12-2. This is the twelfth issue of the AGCIS, which is published to inform the technical staff at MIT/IL and Raytheon about the Apollo guidance computer (AGC) subsystem. In Issue 15 (paragraphs 15-98 through the end) various AGC programs are discussed and their program sections mentioned. This issue is a description of program sections Job Control and Task Control, as used in program SUNRSE33. Previously, these two program sections were called the Executive and the Waitlister. Job Control and Task Control were originally designed for the AGC3 by Dr. J. H. Laning of MIT/IL, modified for the AGC4 by C. A. Muntz, also of MIT/IL, and further modified for program Sunrise by C. A. Muntz.

12-3. An AGC program is defined as the entire content of F memory (paragraph 15-5) and consists of various program sections. Program sections may consist of several major routines, which are composed of minor routines and subroutines.

12-4. A routine, which is assigned a certain priority of execution, is called a Job. The execution of a Job may be requested by another Job or by a Task (Task is defined in paragraph 12-7). When functioning in this capacity, such a Job or Task is referred to as a requesting routine. The requesting routine specifies the address of the requested Job and its priority. A Job, whose execution has been requested, may be in one of two states at any time; either in the active state or in the dormant state. If the Job is in the active state, it is actually being executed or it is awaiting execution. If the Job is in the dormant state, it is not presently being executed and awaiting reactivitation.

12-5. The Job Control schedules the execution of active Jobs according to their priority. Whenever more than one Job is to be executed, the Job Control arranges for the execution of that active Job which has the highest priority. If the execution of a Job is requested which has a priority higher than that of the Job presently being executed, the Job being executed is suspended and the execution of the requested Job is initiated immediately. When the execution of a Job is completed, the execution of the active Job with the next highest priority is initiated or resumed.

12-6. A Job may need information not available yet, or it may have to wait for a certain event to occur. In such cases, the Job requests the Job Control to put it in the dormant state. Any other Job or a Task may request the Job Control to return the dormant Job to the active state at a later time as required. While the dormant Job is in that state, the active Job of the highest priority is executed.

12-7. A routine which must be executed at a specific time is called a Task. The execution of a Task may be requested by a Job, another Task, or itself. The time interval between the execution request and the execution start of a Task cannot be greater than two minutes. The Task Control schedules the execution of Tasks according to their execution start times.

12-8. The address of a program section, a major or a minor routine, a Job, a Task, is the address of the first instruction of that program section, routine, etc. The address of a Job Area, a Work Area, a Pushlist, etc., is the address of the first location of a set of E memory locations. The address of a Job Area location, a Work Area location, a Pushlist location, etc., is the address of that location within a Job Area, Work Area, Pushlist, etc.

12-9. JOB CONTROL

12-10. GENERAL DESCRIPTION

12-11. Program section Job Control (originally named Executive) consists of two major routines: the Load routine and the Swap routine. Table 12-1 lists the execution times of these two routines. The Job Control carries out the following operations:

a. It enters pertinent information concerning a Job, whose execution has been requested, into a Job Area and, if necessary, also into a Work Area. A Job Area consists of 8 locations in E memory; 8 different Job Areas are provided; a Work Area consists of 43 locations in E memory; 5 different Work Areas are provided (see table 15-15). The execution of a Job (paragraphs 12-4 through 12-6) may be requested by another Job or by a Task.

b. It schedules the execution of active Jobs according to their priority.

c. It puts a Job into a dormant state if that Job requests so.

d. It returns a Job to the active state if another Job or a Task requested so.

12-12. When the execution of a Job is requested, the Load routine does the following:

a. It searches for a vacant Job Area and, if necessary, also for a vacant Work Area.

b. It enters pertinent information about the requested Job into a vacant Job Area, and also a vacant Work Area if necessary.

c. It compares the priority of the Job just entered against the priority of the highest priority Job contained in any other Job Area. The number of different levels of priority provided is 31. The priority of a Job is signified by a 5-bit code (octal 01 through 37) in bit positions

14 through 10 of register PRIORITY (table 15-15) used by that Job. The 5-bit code is entered into register PRIORITY by the requesting routine. The Dummy Job is a Job having the lowest priority (01).

d. If the priority of the Job just entered is higher than that of the former highest priority Job, the Load routine enters into register NEWJOB (table 15-15) a tag which specifies the Job Area into which the new Job has been entered. The tag entered into register NEWJOB is a multiple of eight. If the tag is 00000, it indicates that the Job of highest priority is using Job Area 1 and that it is actually being executed. If the tag is not 00000, it indicates that the Job of highest priority uses another Job Area and is awaiting execution. Its execution will be initiated at the next Job breakpoint as explained in the next paragraph. Tag 00010 indicates that the Job of highest priority uses Job Area 2; tag 00020 that it uses Job Area 3, etc., and tag 00070 that it uses Job Area 8.

Table 12-1

EXECUTION TIMES

Routine	MCT's			msec		
	Min	Ave	Max	Min	Ave	Max
Load Routine						
Entry at NOVAC	103	165.5	218	1.205	1.936	2.551
Entry at FINDVAC	99	151.5	204	1.158	1.773	2.387
Entry at JOBWAKE	93	175.5	248	1.088	2.053	2.902
Swap Routine						
Entry at ENDOFJOB	187	249.5	312	2.188	2.919	3.650
Entry at ENDJOB1	182	249.5	307	2.129	2.861	3.592
Entry at JOBSLEEP ⚠	183	245.5	308	2.141	2.872	3.604
Entry at CHANG1	147	149	151	1.720	1.767	1.743
Entry at CHANG2	139	141	143	1.626	1.650	1.673
WAITLIST	90	97.5	105	1.053	1.141	1.228
T3RUPT	86	88	90	1.006	1.030	1.053

⚠ 7 MCT's or 0.082 msec less if program control is returned to NEWORDER of Interpreter

12-13. Jobs written in basic language must be designed such that a Job breakpoint for testing the content of NEWJOB is provided at least every 20 msec (time losses due to interruptions not included) or as often as convenient. The breakpoint that must be inserted in program portions written in basic language is arranged as follows:

```
CCS NEWJOB
TC  CHANG1
```

In the case of Jobs written in interpretive language, register NEWJOB is tested automatically by the Interpreter after the execution of each Interpretive Instruction (on the average every 5 msec). In this case a Job breakpoint is represented by:

```
CCS NEWJOB
TC  CHANG2
```

Whenever the test indicates that the content of NEWJOB is 00000, the execution of the Job being executed continues. Whenever the test indicates that the tag is not 00000, the Swap routine takes control. The Swap routine moves the pertinent information of the highest priority Job into Job Area 1, and moves the pertinent information of the Job previously in Job Area 1 to the Job Area formerly used by the highest priority Job. Whenever the execution of a Job is finished, the Swap routine searches the Job Areas for the Job of next highest priority and enters it into Job Area 1. An example is given in the following paragraph.

12-14. Assume Job A of priority 27 uses Job Area 1, Job B of priority 1 (Dummy Job) uses Job Area 4, Job C of priority 13 uses Job Area 6, the execution of Job D of priority 11 is requested by Job A, and the execution of Job E of priority 15 will be requested by Job C. When the execution of Job D is requested by Job A, the Job Control loads the pertinent Job D information into Job Area 8 (figure 12-1) and then the execution of Job A is continued. When the execution of Job A is completed, the Job Control moves the pertinent Job C information into Job Area 1 and executes Job C. When the execution of Job E is requested by Job C, the Job Control loads the pertinent Job E information into Job Area 7 and then returns program control to continue the execution of Job C. At the next Job breakpoint in Job C, Jobs C and E are swapped, bringing Job E into Job Area 1 and Job C into Job Area 7. Thereafter, Job E is executed. When the execution of Job E has been completed, Job C is brought back into Job Area 1 and its execution is continued. Job D is then executed upon the completion of Job C. After the completion of Job D, Job B is brought into Job Area 1. Job B, the Dummy Job, is a closed loop routine, which stays in Job Area 1 until

a Job breakpoint notifies the Job Control that the execution of a genuine Job (any Job other than the Dummy Job) has been requested. When the

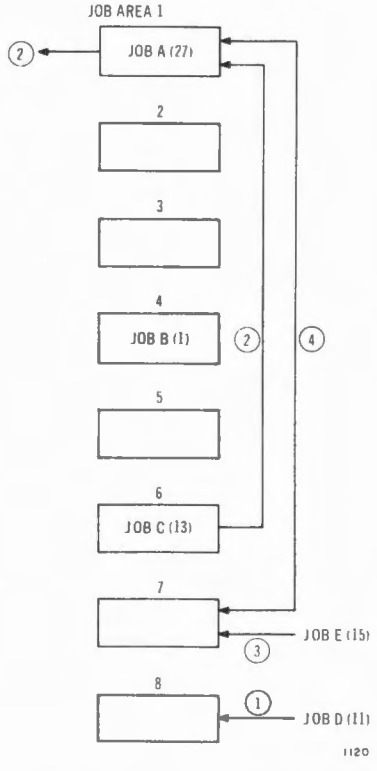

Figure 12-1. Moving of Jobs

execution of a genuine Job is requested, routine Dummy Job moves to another Job Area. If seven Job Areas are occupied by genuine Jobs (one Job Area is always used by the Dummy Job) and the execution of another genuine Job is requested, an alarm is caused. If all Work Areas are occupied and the execution of a Job which needs a Work Area is requested, an alarm is also caused.

12-15. The Dummy Job is a short, closed loop routine which is executed repeatedly as long as it stays in Job Area 1. The Dummy Job is executed alone or with other exercising routines dependent on the content of register SMODE. In either case, the content of register NEWJOB is tested every time the execution of the Dummy Job or the execution of the Dummy Job and the exercising routine is repeated. Whenever register NEWJOB contains a tag other than 00000 at the execution of a Job breakpoint, program control is transferred to the Swap routine to initiate the execution of the requested Job. Whenever

NEWJOB contains tag 00000, the execution of the Dummy Job and the exercising routines (if requested) is continued. The Dummy Job is entered into Job Area 1 the first time during the execution of program section, Fresh Start And Restart, and remains in a Job Area throughout the entire operation of the computer. Dummy Job and Dummy Task keep the AGC subsystem idling as described in paragraph 12-75.

12-16. The execution of a Job can be interrupted in favor of a Job of higher priority as previously discussed in paragraphs 12-12d through 12-14. The execution of a Job can also be interrupted by putting it into a dormant state. The request to put a Job into the dormant state can only come from that Job itself. When a Job requests to be put into the dormant state, the Swap routine of the Job Control complements the content of register PRIORITY used by this Job. Complementing the content of register PRIORITY changes the state of a Job from active to dormant. Consequently, the Swap routine moves the active Job having the highest priority into Job Area 1 for its execution; and the dormant Job from Job Area 1 into the Job Area from which the active Job was just taken.

12-17. A Job, which is in the dormant state, can be returned to its active state upon request of a Task or another Job. This is referred to as a reactivating routine. In this case, the Load routine of the Job Control searches for the Job Area used by the Job to be reactivated and recomplements the content of register PRIORITY. Recomplementing the content of register PRIORITY returns the state of that Job from the dormant to the active state. If the reactivated Job becomes the active Job of highest priority, it will be returned to Job Area 1 at the next Job breakpoint. Otherwise the Job has to await execution until it becomes the active Job of highest priority.

12-18. LOAD ROUTINE

12-19. Figure 12-2 illustrates the functional flow of the Load routine, which may be entered at location NOVAC, FINDVAC, or JOBWAKE. A detailed flow chart of the Load routine is shown in attachment 12-1.

12-20. CALLING SEQUENCES

12-21. If the Job whose execution has been requested requires only the use of a Job Area (as do many Jobs which are written in basic machine language only), the Load routine is entered at NOVAC by

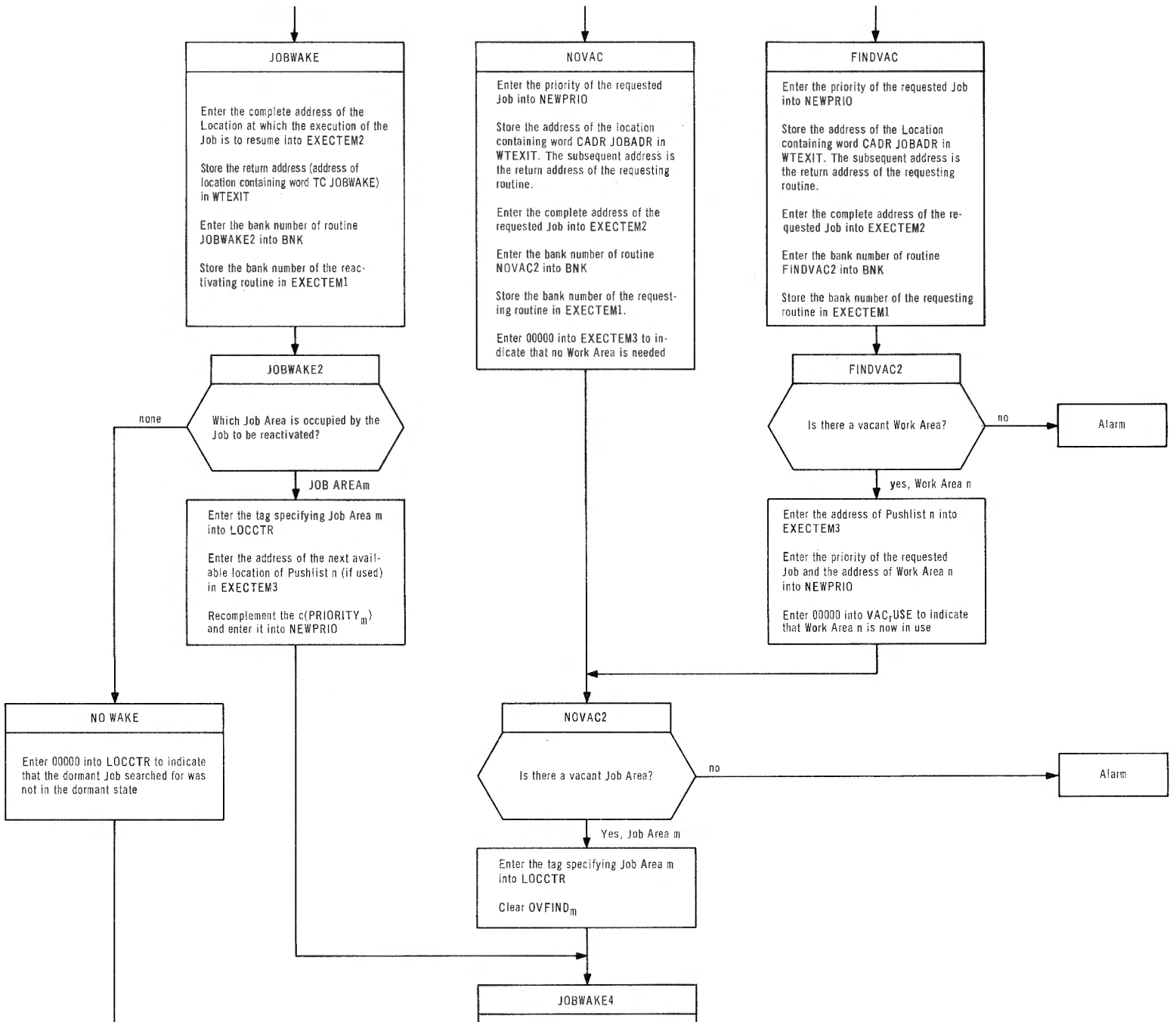

Figure 12-2. Load Routine Functional Flowchart

means of the following calling sequence which is a part of the requesting routine:

```
CAF   JOBPR
TC    NOVAC
CADR  JOBADR
```

The symbol JOBPR is used as an example and represents the location within the requesting routine which specifies the priority of the requested Job. The priority words contain octal quantities 01 through 37 in bit positions 14 through 10 and ZERO's in the other bit positions. The symbol JOBADR is used as an example and CADR JOBADR represents the complete address of the requested Job. After entering the pertinent Job information into a Job Area, the Load routine returns control to the return location (location immediately following the Job address in the calling sequence).

12-22. If the requested Job requires the use of a Job Area and a Work Area (as do all Jobs which are written in interpretive language and some Jobs which are written in basic machine language), the Load routine is entered at FINDVAC by means of the following calling sequence which is also a part of the requesting routine:

```
CAF   JOBPR
TC    FINDVAC
CADR  JOBADR
```

12-23. If a Job is to be returned from the dormant state to the active state, the Load routine is entered at JOBWAKE by means of the following calling sequence which is part of the reactivating routine:

```
CAF   WAKEADR
TC    JOBWAKE
```

If the reactivating routine is not in the basic mode, as assumed above, but in the interpretive mode, the calling sequence must be preceded by Interpretive Instruction EXIT, and it must be followed by instruction TC INTPRET if the execution of the Job is to continue in the interpretive mode. The symbol WAKEADR is used as an example and represents a 12-bit address. This addressed location contains the complete address of the instruction (Regular Instruction or Interpretive Instruction Word) where the execution of the dormant Job is resumed. If the reactivation of a dormant Job is requested and the resume address of this Job cannot be found in any of the Job Areas, program control is returned to the requesting (reactivating) routine.

12-24. The above calling sequences, whether specifying transfer to NOVAC, FINDVAC, or JOBWAKE, must be preceded by an INHINT and followed by a RELINT in order to prevent the interruption of the loading operations. If the requesting or reactivating routine is a Job, the programmer has to supply the instruction INHINT (before CAF JOBPR or CAF WAKEADR, or right after it) and the instruction RELINT (after CADR JOBADR or TC JOBWAKE). If the requesting or reactivating routine is a Task, instructions RPT (caused by overflow of TIME3) and RSM (at the end of routine TASKOVER) cause INHINT and RELINT automatically (refer to paragraphs 2-100 and 2-103).

12-25. ENTRY AT NOVAC

12-26. When the Load routine is entered at location NOVAC (in bank 01), the priority of the requested Job is stored in bit positions 14 through 10 of register NEWPRIO (table 15-15). The Load routine then stores the address of the location containing the word JOBADR of the calling sequence in register WTEXIT. The complete address of the requested Job is stored in register EXECTEM 2. Then the bank code 04 is entered into register BNK to switch to bank 04 where minor routine NOVAC2 is located. The bank number of the requesting routine is stored in register EXECTEM 1. Finally, 00000 is entered into register EXECTEM 3 to indicate that the requested Job does not need a Work Area.

12-27. Minor routine NOVAC2 searches for a Job Area not in use. Each of the eight Job Areas is composed of 8 storage registers (refer to table 15-15). Register PRIORITY of a Job Area contains 77777 if its Job Area is available. When a Job Area is in use, register PRIORITY contains the priority of the Job using this Job Area (paragraph 12-12c). Job Area 1 is always used by the active Job of highest priority or by the Dummy Job if no active Job is present in any Job Area. For this reason the search for an available Job Area starts at Job Area 8 and proceeds to Job Area 2; Job Area 1 is never tested. Routine NOVAC2 examines the content of register PRIORITY of Job Areas 8 through 2 until one is found which contains 77777. If the test discloses that all Job Areas are in use, an alarm is caused. At the end of the search, register LOCCTR contains octal quantity 70, 60, 50, 40, 30, 20, or 10 dependent upon whether Job Area 8, 7, 6, 5, 4, 3, or 2 is available; it contains 0 if no Job Area is available. Register OVFIND of the available Job Area is then cleared (set to 00000).

12-28. Minor routine JOBWAKE4 transfers the priority of the requested Job from register NEWPRIO to register PRIORITY_m, m being the tag which indicates the number of the available Job Area. Next, the

bank number of the requested Job is stored in register $PUSHLOC_m$. Thereafter the 12-bit address (as entered into register S for addressing memory, table 15-3) of the requested Job is calculated and stored in register $IWLOC_m$.

12-29. So far, the Load routine specified the Job Area which should be used by the requested Job and entered pertinent information of this Job into certain registers. Now the priority of the requested Job is compared with the priority of the other active Jobs to determine whether or not the execution of the requested Job should be initiated at the next Job breakpoint. If the priority of the requested Job is higher than the priority of any other active Job, a tag (a multiple of 8, paragraph 12-12d) is entered into register NEWJOB, indicating that Job Area m now contains the active Job of highest priority. If the priority of the requested Job is not higher, the content of NEWJOB is left unchanged.

12-30. Note that prior to the occurrence of a Job breakpoint in the Job using Job Area 1, two or more successive Job requests could have been made and information entered in two or more Job Areas. Furthermore, all newly entered Jobs might be of higher priority than the Job using Job Area 1. When this occurs, the priority comparison test ensures that the tag specifying the Job Area used by the active Job of highest priority is entered into register NEWJOB. For further details refer to attachment 12-1.

12-31. Finally, minor routine NOWAKE2 restores the bank number of the requesting routine in register BNK and returns program control to the requesting routine in order to resume its execution.

12-32. ENTRY AT FINDVAC

12-33. When the Load routine is entered at location FINDVAC (in bank 01), it operates the same as if it were entered at NOVAC except that a Work Area must also be allocated. The address of the assigned Work Area is stored in the Job Area to be used. First the Load routine loads registers NEWPRIO, WTEXIT, EXECTEM2, BNK, and EXECTEM1 as described in paragraph 12-25.

12-34. Minor routine FINDVAC2 searches for a Work Area not in use. Each of the five Work Areas is composed of 43 storage registers (refer to table 15-15) and each Work Area is preceded by a VACrUSE register (r=1, 2, 3, 4, or 5). Each VACrUSE register contains its own address if its Work Area is available, and the quantity 00000 if its Work Area is in use. The VACrUSE register addresses are entered

into these registers the first time by program section Fresh Start And Restart. Each Work Area contains a Pushlist occupying the first 32 locations of that Work Area. (A Pushlist is a series of registers reserved for temporary storage, table 15-15.) The search for an available Work Area is simply a matter of testing the content of registers VACrUSE, starting with VAC1USE, until one is found containing some value other than zero. If the test discloses that Work Area n is available, the address of Work Area n is entered into register EXECTEM 3. If no Work Area is available, an alarm is caused. After locating an available Work Area, the priority of the requested Job and the address of the available Work Area (which is also the address of a Pushlist) are "packed" into register NEWPRIO. "Packing" means storing two distinct quantities into one register without distorting either. In the case of register NEWPRIO, the priority is stored in bit positions 14 through 10 and the address n in bit positions 9 through 1. After register NEWPRIO has been packed, the content of register VACrUSE is set to 00000 indicating Work Area n is in use.

12-35. Minor routines NOVAC2 and JOBWAKE4 operate as described in paragraphs 12-27 through 12-31, except that register $PRIORITY_m$ now also contains the address of a Work Area as well as the priority. In addition, the address of the Pushlist is packed, together with the bank number of the requested Job, in register $PUSHLOC_m$. During the execution of a Job, the address in register PUSHLOC is changed whenever data is entered into or read out of the Pushlist. Therefore bit positions 10 through 1 of this register always contains the address of the first available, or empty, register of that Pushlist.

12-36. ENTRY AT JOBWAKE

12-37. When the Load routine is entered at location JOBWAKE (in bank 01), it searches the Job Areas for the Job to be reactivated, changes the state of this Job, and schedules the execution of the reactivated Job. First, the complete address of the location where the execution of the Job is to be reactivated is stored in register EXECTEM2. The address of instruction TC JOBWAKE of the calling sequence is stored in register WTEXIT. The bank code 04 is entered into register BNK to switch to bank 04 where minor routine JOBWAKE2 is located. The bank number of the reactivating routine is stored in register EXECTEM1.

12-38. Minor routine JOBWAKE2 searches for a Job Area containing a dormant Job by examining the content of register PRIORITY of Job Area 8 through 2 until one is found which contains a negative quantity. Once a PRIORITY register has been found that contains a negative

quantity, routine JOBWAKE2 checks whether the resume address contained in register EXECTEM2 agrees with the resume address contained in register IWLOC of the located Job Area. If no agreement exists, routine JOBWAKE2 searches for the next PRIORITY register containing a negative quantity and checks again for address agreement. This operation is repeated until the Job Area is found, which is used by the Job to be reactivated. If that Job cannot be found, 00000 is entered into register LOCCTR indicating the dormant Job searched for was not dormant; program control is returned to minor routine JOBWAKE1 and the reactivating routine. At the end of the successful search, register LOCCTR contains octal quantity 70, 60, 50, 40, 30, 20, or 10 dependent upon whether Job Area 8, 7, 6, 5, 4, 3, or 2 is used by the Job to be reactivated; it contains 0, if the search failed. For details of routine JOBWAKE2, refer to attachment 12-1.

12-39. Before program control is transferred to minor routine JOBWAKE4, the address of the next available Pushlist m location (as contained in register $PUSHLOC_m$) is entered into register EXECTEM3. Finally, the content of register $PRIORITY_m$ is recomplemented; this returns the Job occupying Job Area m to its active state as described in paragraph 12-17.

12-40. Minor routine JOBWAKE4 operates as described in paragraphs 12-28 through 12-31 and 12-35, except that resume addresses are loaded into registers $PUSHLOC_m$ and $IWLOC_m$ instead of the first location of a Pushlist and of a Job.

12-41. SWAP ROUTINE

12-42. Figure 12-3 illustrates the functional flow of the Swap routine, which may be entered at location ENDOFJOB, ENDJOB1, CHANG1, CHANG2, or JOBSLEEP. A detailed flow chart of the Swap routine is shown in attachment 12-2.

12-43. CALLING SEQUENCES

12-44. If the execution of a Job which ends with Basic Instructions has been completed, the Swap routine is entered at ENDOFJOB by means of instruction TC ENDOFJOB. Instruction TC ENDOFJOB must be the last instruction of all Jobs which terminate in the basic mode. If a Job terminates in the interpretive mode, that is with Interpretive Instructions, the Swap routine is entered at ENDJOB1 by means of the Interpretive Instruction, RTB ENDJOB1. Instruction RTB ENDJOB1 must be the last instruction of an Interpretive Job. Whenever the Swap routine is entered at ENDOFJOB or ENDJOB1, it first searches all

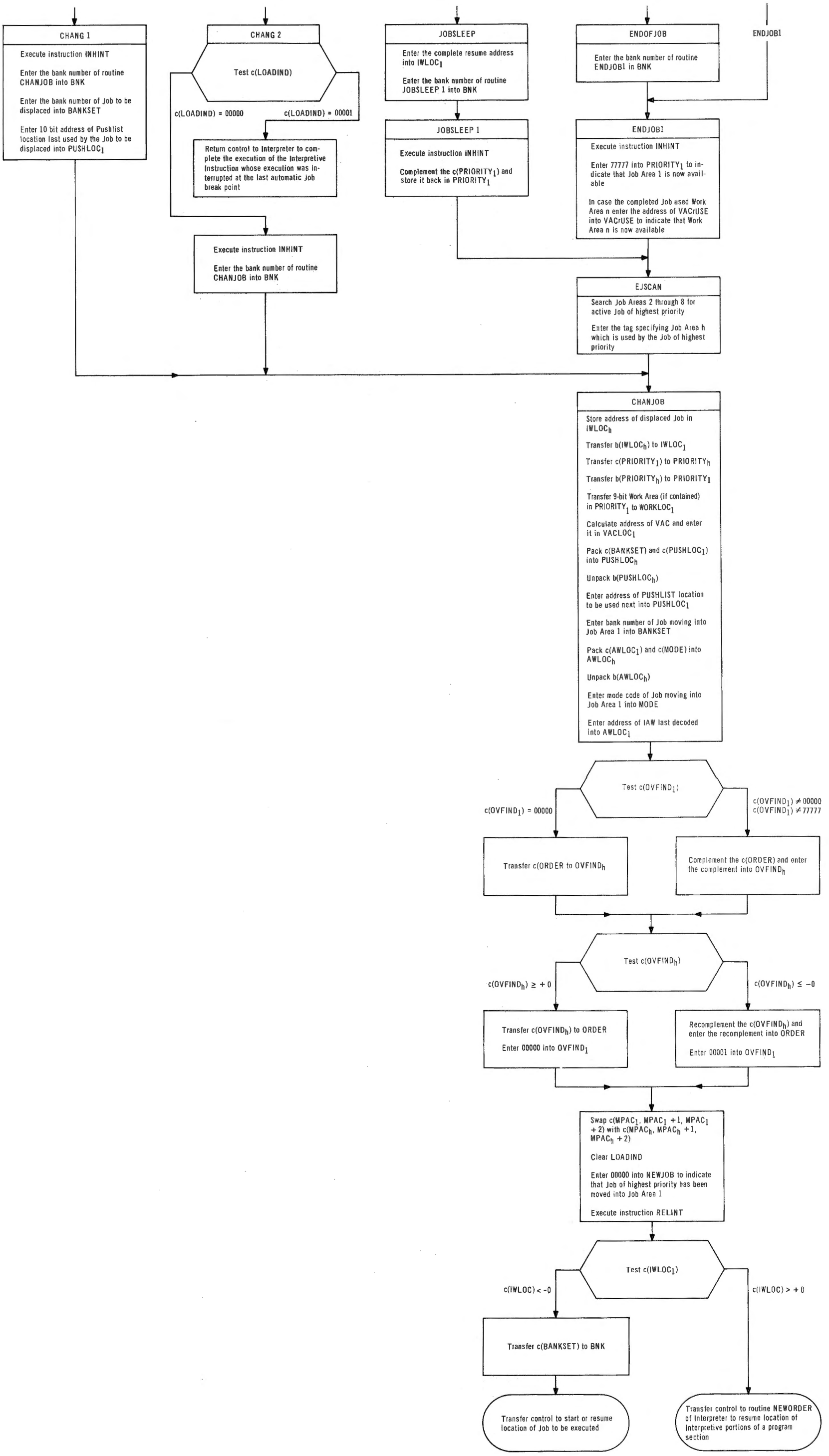

Figure 12-3. Swap Routine Functional Flowchart

CONFIDENTIAL

Job Areas for the active Job of the next highest priority before swapping Jobs.

12-45. If, during the execution of a Job, a Job breakpoint (paragraph 12-13) is encountered and register NEWJOB does not contain 00000, it means that a Job of higher priority has been entered since the last Job breakpoint and the Job must be executed immediately. Therefore, program control is transferred to CHANG1 or CHANG2. The Swap routine is entered at CHANG1 only from a sequence of Regular Instructions which contains a Job breakpoint. The Swap routine is entered at CHANG2 only during the execution of Interpretive Instructions. Whenever the Swap routine is entered at CHANG1 or CHANG2, it does not have to search for the Job to be executed next because the Job Area used by the highest priority Job is specified by the content of register NEWJOB.

12-46. If, during execution, a Job requests to be put into the dormant state, the Swap routine is entered at JOBSLEEP by the following calling sequence which is within the Job itself:

```
CAF  WAKEADR
TC   JOBSLEEP
```

If the Job requesting to be put into the dormant state is not in the basic mode, as assumed above, the calling sequence must be preceded by Interpretive Instruction EXIT and must be followed by instruction TC INTPRET if the execution of the Job is to continue in the interpretive mode. Compare with paragraph 12-23.

12-47. ENTRY AT ENDOFJOB

12-48. When the Swap routine is entered at location ENDOFJOB (in bank 01), it enters the bank code 04 into register BNK to switch to bank 04 where minor routine ENDJOB1 is located.

12-49. Minor routine ENDJOB1 starts with the execution of instruction INHINT which prevents any interruption of the search and swap operations. The content of register $PRIORITY_1$ (used by the completed Job) is then set to 77777 (priority minus zero) to indicate that the Job has been completed and that Job Area 1 is now available. Next, the former content of bit positions 9 through 1 of register $PRIORITY_1$ is tested to determine whether the completed Job also used a Work Area. If these bit positions contained the Work Area address, then the VACrUSE register for that Work Area is loaded with the VACrUSE address to indicate that the Work Area is now available too. If these bit positions contained ZERO's, control is transferred directly to minor routine EJSCAN.

12-50. Minor routine EJSCAN searches for the active Job of next highest priority. This is accomplished by testing and comparing the contents of the PRIORITY registers in Job Areas 2 through 8. Assuming that the search discloses that Job Area h now contains the Job of highest priority, then the tag 10 (h-1) is entered into register NEWJOB to indicate that the Job of highest priority is presently using Job Area h. For details of the search operation, see attachment 12-2. After the search for the active Job of next highest priority, program control is transferred to minor routine CHANJOB. At this time register A, the registers of Job Area 1, and the Work Area if one was used, contains irrelevant information providing the Swap routine was entered at ENDOFJOB. The operation of routine CHANJOB, which swaps Jobs, is described in paragraphs 12-63 through 12-68.

12-51. ENTRY AT ENDJOB1

12-52. The Swap routine is entered directly at location ENDJOB1 (in bank 04) whenever the execution of a Job ending with Interpretive Instruction has been completed (paragraph 12-44). Switching from bank 03, where the Interpreter is located, to bank 04, where routine ENDJOB1 is located, is carried out by routine SWCALL of program section, Interbank Communication, on request of the Interpreter. For operation of minor routines ENDJOB1 and EJSCAN, refer to paragraphs 12-49 through 12-50. When the execution of routine EJSCAN is completed, register A, the registers of Job Area 1, and the Work Area again contain irrelevant information. Program control is transferred to minor routine CHANJOB described in paragraphs 12-63 through 12-68.

12-53. ENTRY AT JOBSLEEP

12-54. When the Swap routine is entered at location JOBSLEEP (in bank 01), it enters into register $IWLOC_1$ the complete address of the location where the routine should resume execution. This occurs when the dormant Job is reactivated and entered into register $IWLOC_1$. The bank code 04 is then entered into register BNK to switch to bank 04, where minor routine JOBSLEEP1 is located.

12-55. Minor routine JOBSLEEP1 first executes instruction INHINT which prevents any interruption of the search and swap operation. Then the content of register $PRIORITY_1$ is complemented, thus putting the Job occupying Job Area 1 into the dormant state as described in paragraph 12-16.

12-56. Minor routine EJSCAN searches for the active Job of the next higher priority as described in paragraph 12-50. When the execution of routine EJSCAN is completed, register A contains the resume address which was entered into register $IWLOC_1$ earlier (paragraph 12-54). Finally, program control is transferred to routine CHANJOB described in paragraphs 12-63 through 12-68.

12-57. ENTRY AT CHANG1

12-58. When the Swap routine is entered at location CHANG1 (in bank 01) instruction INHINT is executed immediately to prevent any interruption of the swap operation. Then the bank number of minor routine CHANJOB is set into register BNK. The bank number of the Job being interrupted and displaced from Job Area 1 is transferred to register BANKSET. Register $PUSHLOC_1$ is unpacked and the information contained in bit positions 10 through 1 is preserved while the content of bit positions 15 through 11 is discarded. (Only bit position 10 through 1 of register PUSHLOC may be used by a basic Job for storing any information other than the address of the current Pushlist register). The return address (the address of the first instruction to be executed when the execution of the Job being displaced is resumed) is complemented and stored in register A. This address is stored in its complemented form to indicate that the Job was in the basic mode at the time of displacement. Finally, program control is transferred to minor routine CHANJOB described in paragraphs 12-63 through 12-68.

12-59. ENTRY AT CHANG2

12-60. Interpretive Instructions can only be decoded and executed by the Interpreter. During operation, the Interpreter tests the content of register NEWJOB. This test ordinarily occurs upon the completion of decoding and execution of most Interpretive Instructions, at which time LOADIND contains 00000. However, during the execution of some instructions (many Dual Quantity Instructions), the content of register NEWJOB is tested before the instruction has been completely executed, in this case LOADIND contains 00001. The automatic Job breakpoint in the Interpreter tests the content of NEWJOB; if the content is a non-zero tag, the breakpoint transfers control immediately to CHANG2 without even considering the content of LOADIND. It is for this reason that minor routine CHANG2 has to test the content of LOADIND first and return control to the Interpreter in case the execution of an Interpretive Instruction has not been completed.

12-61. When the Swap routine is entered at location CHANG2 (in bank 01), the content of register LOADIND is tested. If the test discloses that register LOADIND contains 00000, instruction INHINT is executed to prevent any interruption of the swap operation. The bank number of minor routine CHANJOB (i. e. 04) is set into register BNK. The address of the last decoded and executed Interpretive Instruction Word of the Job being displaced is stored in its noncomplemented form in register A. The address is stored in its noncomplemented form to indicate that the Job was in the interpretive mode at the time of displacement. Finally, program control is transferred to minor routine CHANJOB described in paragraphs 12-63 through 12-68.

12-62. If the test discloses that register LOADIND contains 00001, control is returned to the Interpreter to allow it to complete the execution of the Interpretive Instruction being executed. Thereafter, the Swap routine is entered at CHANG2 again and is now able to perform the operation described for the case when LOADIND contains 00000.

12-63. ROUTINE CHANJOB

12-64. No matter at which point (ENDOFJOB, ENDJOB1, CHANG1, CHANG2 or JOBSLEEP) the Swap routine has been entered when minor routine CHANJOB is entered, an active Job using any Job Area 2 through 8 has a higher priority than the Job using Job Area 1. For convenience, the Job Area used by the highest priority active Job at the entry of CHANJOB is designated Job Area h. The priority of the Job being displaced from Job Area 1 may be minus zero because its execution has been completed; it may be positive, but less positive than the priority of the active Job of highest priority; or it may be negative which indicates a dormant Job. In any case, the highest priority Job is moved from Job Area 1 by routine CHANJOB, and the Job occupying Job Area 1 is moved to Job Area h.

12-65. When entering minor routine CHANJOB, Job Area 1 may be used by a Job just completed, a Job just interrupted, or a Job just put into the dormant state. In the first case, register $PRIORITY_1$ contains 77777; all other registers of $Job\ Area_1$ contain irrelevant information. In the second case, register $PRIORITY_1$ contains the priority of the interrupted Job and the address of a Work Area, if any; register $PUSHLOC_1$ contains the bank number of this Job and the address of a Pushlist location, if any; register A contains the complemented address of a Regular Instruction to be executed next or the noncomplemented address of the Interpretive Instruction Work last executed. In the third case, registers $PRIORITY_1$ and $PUSHLOC_1$ contain the same information as in the second case, except that the content of register PRIORITY

has been complemented; register A contains the noncomplemented address of a Regular Instruction or Interpretive Instruction Word to be executed next.

12-66. When entering minor routine CHANJOB, Job Area h is used by the active Job of highest priority. The execution of this Job may be taking place for the first time, or it might have been started and interrupted previously. In the first case, register $PRIORITY_h$ contains the priority of the displacing Job and a Work Area address (if needed). In addition, register $PUSHLOC_h$ contains the bank number of this Job and a Pushlist address (if any); and register $IWLOC_h$ contains the complemented address of this Job (the first instruction of an interpretive Job is the basic instruction TC INTPRET). In the second case, register $PRIORITY_h$ contains the priority of the displacing Job and a Work Area address (if any). Also register $PUSHLOC_h$ contains the bank number of this Job and the address of a Pushlist location (if any), and register IWLOC contains the complemented address of the Regular Instruction to be executed next or the noncomplemented address of the Interpretive Instruction Word last executed.

12-67. Whenever the Swap routine has been entered at CHANG1 or CHANG2, register $IWLOC_h$ can only contain the starting address of a Job when entering CHANJOB and never a return address because entrance at CHANG1 or CHANG2 occurs only if a higher priority Job has been loaded since the execution of the last Job breakpoint (paragraph 12-45). Whenever the Swap routine has been entered at ENDOFJOB or ENDJOB1, register $IWLOC_h$ may contain a start or a resume address. This occurs because the Job to be executed next can be a Job being started for the first time or a Job being resumed. Only the address of an Interpretive Instruction Word executed last is stored in register IWLOC in its noncomplemented form upon entry into CHANJOB; the Interpreter increments this address thus providing the resume address. The start address of a basic Job or of an Interpretive Job, as well as the address of the Regular Instruction to be executed next, are stored in their complemented forms. This arrangement is necessary so that CHANJOB can test to determine whether the first instruction to be executed for the Job in Job Area h is a Regular Instruction (TC INTPRET included) or in Interpretive Instruction.

12-68. After entering minor routine CHANJOB, the following steps (See attachment 12-2 for details) are taken to swap information and complete the operation:

a. A return address, or an irrelevant address, if the execution of the Job has been completed, is transferred

from register A to register $IWLOC_h$. The address contained in $IWLOC_h$ is taken into register A and is transferred to register $IWLOC_1$.

b. The content of register $PRIORITY_1$ is brought into register A and transferred to register $PRIORITY_h$ while the content of register $PRIORITY_h$ is transferred to register $PRIORITY_1$. Register $PRIORITY_1$ now contains the priority and a Work Area address (if any). Both pertain to the highest priority Job.

c. The 9-bit Work Area address (if any) contained in register $PRIORITY_1$ is entered into register $WORKLOC_1$.

d. The address of the VAC (if used) is calculated by adding the octal quantity 40 to the Work Area address and the VAC address is stored in register $VACLOC_1$. If the Job being transferred to Job Area 1 does not need a Work Area, register $WORKLOC_1$ contains 00000 and register VACLOC contains 00040.

e. The content of register BANKSET (bank number of the Job being removed from Job Area 1) and the content of register $PUSHLOC_1$ (address of a Pushlist location, if any) are packed into register $PUSHLOC_h$. The former content of register $PUSHLOC_h$ is brought into register A.

f. The content of register A, which pertains to the highest priority Job, is unpacked. The address of a Pushlist location (if any is used by the Job being moved into Job Area 1) is stored in register $PUSHLOC_1$ and the bank number of the Job being entered is stored in register BANKSET. If the Job being transferred to Job Area 1 does not need a Work Area, register $PUSHLOC_1$ contains 00000.

g. The content of register $AWLOC_1$ (address of IAW decoded last by the Interpreter, if only the Job is an interpretive Job, otherwise 00000) is shifted two places to the left. The shifted quantity and the content of register MODE Interpreter mode are packed into register $AWLOC_h$. The former content of register $AWLOC_h$ is brought into register A and stored in register $AWLOC_1$.

h. The content of register A, which pertains to the Job of highest priority, is unpacked. The mode code is stored in register MODE, and the IAW address is stored (after

shifting twice to the right) in register $AWLOC_1$. If the Job being transferred to Job Area 1 is not an interpretive Job, register MODE and register $AWLOC_1$ contain irrelevant information.

j. If register $OVFIND_1$ contains 00000, the content of register ORDER (see table 15-15) is transferred to register $OVFIND_h$. (Register OVFIND is used only by the Interpreter during the execution of Interpretive Instructions.) If register $OVFIND_1$ contains a positive or a negative quantity, the complemented content of register ORDER is transferred to register $OVFIND_h$. As information is transferred to register $OVFIND_h$, its former content is taken into the CP and tested. If register $OVFIND_h$ contained 00000 or a positive quantity, this quantity is entered into register ORDER and register $OVFIND_1$ is set to 00000. If register $OVFIND_h$ contained 77777 or a negative quantity, its content is recomplemented and entered into register ORDER and register $OVFIND_1$ is set to 00001.

k. The contents of the MPAC in Job Area 1 and the MPAC in Job Area h are swapped.

l. Register LOADIND is cleared. This is necessary because the register is also used by other routines and must be set to 00000 before the execution of Interpretive Instruction.

m. The content of register NEWJOB is set to 00000 to indicate that the Job of highest priority is now occupying Job Area 1.

n. The instruction RELINT is executed to again allow program interruptions.

p. Finally, the content of register $IWLOC_1$ is tested. If register $IWLOC_1$ contains a positive quantity (address of IIW last executed) program control is transferred to minor routine NEWORDER of the Interpreter in order to execute the next Interpretive Instruction and thus resume the execution of the interpretive Job. If register $IWLOC_1$ contains a negative quantity (complemented address of a basic Job or of an interpretive Job, or complemented address of a Regular Instruction to be executed, the bank number of the Job is entered into register BNK and program control is transferred to the Job in order to execute that Job, or to resume the execution of that Job.

12-69. TASK CONTROL

12-70. GENERAL DESCRIPTION

12-71. Program section Task Control (previously referred to as Waitlister) consists of two major routines: the WAITLIST routine and the T3RUPT routine. Table 12-1 lists the execution times of these two routines. The WAITLIST routine carries out the following operations:

a. It determines the time when the execution of a requested Task shall be initiated. The execution of a Task (paragraph 12-7) may be requested by a Job, another Task, or itself.

b. It schedules the execution of the requested Task according to its times of initiation. This is accomplished by entering pertinent information of the requested Task into the proper place in the Waitinglist.

The T3RUPT routine carries out the following operations:

a. It initiates the execution of the Task due now.

b. It moves the Tasks whose execution is due next into place 1 of the Waitinglist; and moves the remaining Tasks up one place. Tasks located in places 2 through 6 are moved into places 1 through 5 of the Waitinglist, and Dummy Task is entered into place 6.

12-72. The Waitinglist consists of the Timelist and the Addresslist as shown in table 12-2. Refer to table 15-15 also. The Timelist consists of counter TIME 3 and E memory locations LST1 through LST1 +4. Locations LST1 through LST1 +4 store time quantities $(-\Delta t_2+1)$ through $(-\Delta t_6+1)$ and counter TIME 3 stores the quantity $(040000 - \Delta t_1)$. (In some papers, the expression 1.0 is used instead of octal quantity 040000 to represent the overflow condition of the accumulator A.) The Addresslist consists of E memory locations LST2 through LST2 +5 which store Task addresses.

12-73. The value $\Delta t_1 = T_1 - T_0$ represents the time interval between time T_1, at which the execution of Task 1 shall be initiated, and the time T_0 of the present moment. The value $\Delta t_2 = T_2 - T_1$ represents the time interval between time T_2, when the execution of Task 2 shall

TABLE 12-2

WAITINGLIST CONTENTS

Timelist	Addresslist
c(TIME3) = 040000 $-\Delta t_1$	c(LST2) = Complete Address of Task 1
c(LST1) = $-\Delta t_2 + 1$	c(LST2 +1) = Complete Address of Task 2
c(LST1 +1) = $-\Delta t_3 + 1$	c(LST2 +2) = Complete Address of Task 3
c(LST1 +2) = $-\Delta t_4 + 1$	c(LST2 +3) = Complete Address of Task 4
c(LST1 +3) = $-\Delta t_5 + 1$	c(LST2 +4) = Complete Address of Task 5
c(LST1 +4) = $-\Delta t_6 + 1$	c(LST2 +5) = Complete Address of Task 6

be initiated, and time T_1 when the execution of Task 1 shall be initiated. In general, $\Delta t_n = T_n - T_{n-1}$ represents the time interval between two successive Task execution initializations. The octal quantity 00001 represents 10 msec which is equal to the time intervals at which the time counters are incremented. The largest time quantity that can be stored in an E memory location is 37777, which represents 163.83 sec. The overflow condition of accumulator A is quantity 040000 which represents 163.84 sec. Two minutes is represented by the octal quantity 27340. Thus (040000 - Δt_n) = 10440 for Δt_n = 27340.

12-74. When the operation of the computer is started, the Dummy Task, (i.e. address and Δt of that Task) is entered into places 2 through 6 of the Waitinglist during the execution of program section Fresh Start. The address of the Dummy Task is also entered into place 1 of of the Addresslist and 37777 is entered into counter TIME3. The Dummy Task is located at SVCT3 and consists of only one instruction.

TC TASKOVER

The quantity ($-\Delta t_d$+1) entered for a Dummy Task is always the octal quantity 57777 or -20000 which represents -81.92 sec. This means that Δt_d = 81.93 sec and that a Dummy Task is executed once every 81.93 sec if no genuine Task is in the Waitinglist. If places 2 through 6 are loaded with Dummy Tasks, the total time interval represented by the five $\Delta t_{d's}$ is about seven minutes.

12-75. The purpose of the Dummy Task and the Dummy Job is to keep the AGC subsystem idling until the AGE system or an astronaut requires

computer participation in an operation. Routine DUMMYJOB always occupies one of the eight Job Areas, and the Dummy Task may occupy none, several, or all places in the Waitinglist.

12-76. When a routine requests the execution of a Task, it supplies the address of the requested Task and the time interval Δt_R. The time interval $\Delta t_R = T_R - T_0$ of a requested (genuine) Task represents the time interval between the time T_R when the execution of the requested Task R shall be initiated and to the time of the execution request. As the execution of a Task is requested, the WAITLIST routine determines the place in the Waitinglist into which the requested Task has to be entered, computes a new set of values for the quantities $(040000 - \Delta t_1)$ and $(-\Delta t_2 + 1)$ through $(-\Delta t_6 + 1)$ as necessary, and rearranges the Timelist and the Addresslist accordingly.

12-77. The initiation of the execution of a Task is caused by the overflow of counter TIME3. The content of counter TIME3 is incremented every 10 msec by a signal from Scaler A similar to the way counter TIME1 is incremented (table 15-5 and figure 15-8). Since register TIME3 stores the time quantity $(163.84 \text{ sec} - T_1 + T_0)$ and because T_0 becomes 10 msec larger every 10 msec, overflow occurs at the moment $T_0 = T_1$. At the instant counter TIME3 overflows, the actions described in table 15-5 are initiated. Signal RP1 is generated in the Interrupt Priority Control, instruction RPT and Transfer Routine TIME3RPT are executed, and program control is transferred to routine T3RUPT. Routine T3RUPT moves the Tasks presently located in places 2 through 6 into places 1 through 5 of the Waitinglist, enters a Dummy Task into places 6, and transfers program control to the Task due as indicated in figure 12-4. After the execution of the Task, control is returned to routine T3RUPT, and the interrupted Job if the execution of no other Task is due.

12-78. On scale A of figure 12-5, the conditions are shown which exist when Dummy Task only has been entered into Waitlist (i.e. no genuine Task has been entered yet). Address SVCT3, symbolized by Z, is stored in each location of the Addresslist. The time intervals Δt_2 through Δt_6, each being equal to a Δt_d of 81.93 sec, are stored in locations LST1 through LST1 + 4 of the Timelist. The time interval Δt_1 is the interval between now (T_0) and the execution start (T_1) of a Dummy Task. The start times T_1 through T_6 were established as the operation of the computer was started. The interval Δt_1 decreases as time passes (T_0 increases) and the content $(040000 - \Delta t_1)$ of counter TIME3 is incremented every 10 msec. At the instant Δt_1 becomes zero, counter TIME3 overflows, control goes to T3RUPT, and the execution of the first Dummy Task is initiated.

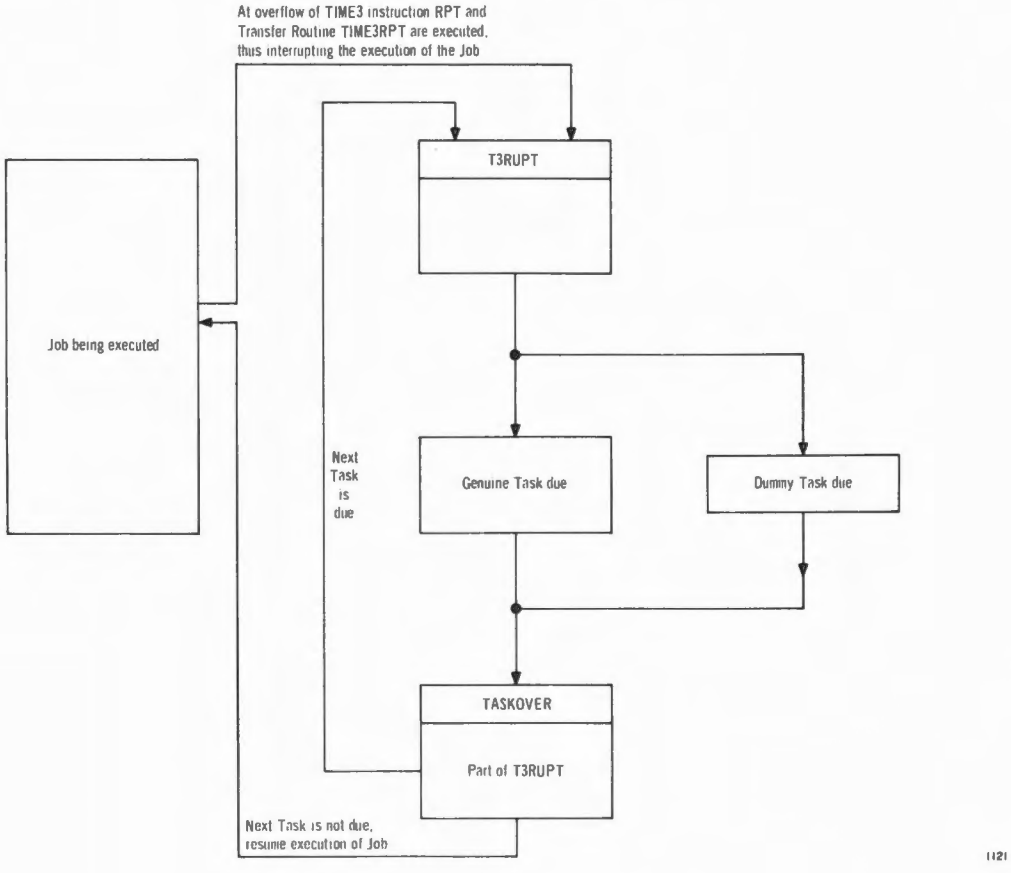

Figure 12-4. Execution of a Task

12-79. Assume a Job requests the execution of Task A at Δt_A = 20 sec from the present which is about 40 sec after computer start. As indicated on scale B of figure 12-5, the execution of Task A should start at time T_A. When the execution of Task A is requested, routine WAITLIST computes the new Δt's, (Δt_1 and Δt_2) which are placed into counter TIME3 and location LST1 according to table 12-2. The former time intervals, Δt_2 through Δt_5 (scale A), are moved into locations LST1 + 1 through LST + 4 and become the new Δt_3 through Δt_6 (scale B). The former time interval Δt_6 (scale A) is pushed out and discarded. Similarly, the complete addresses of Task A are entered into location LST2, and the complete addresses of the Dummy Tasks are pushed down one place and one address is pushed out.

12-80. Furthermore, assume that about 55 sec after the computer has been started a Job requests the execution of Task B at Δt_B = 120 sec (maximum Δt for a genuine task) from now. As indicated on scale

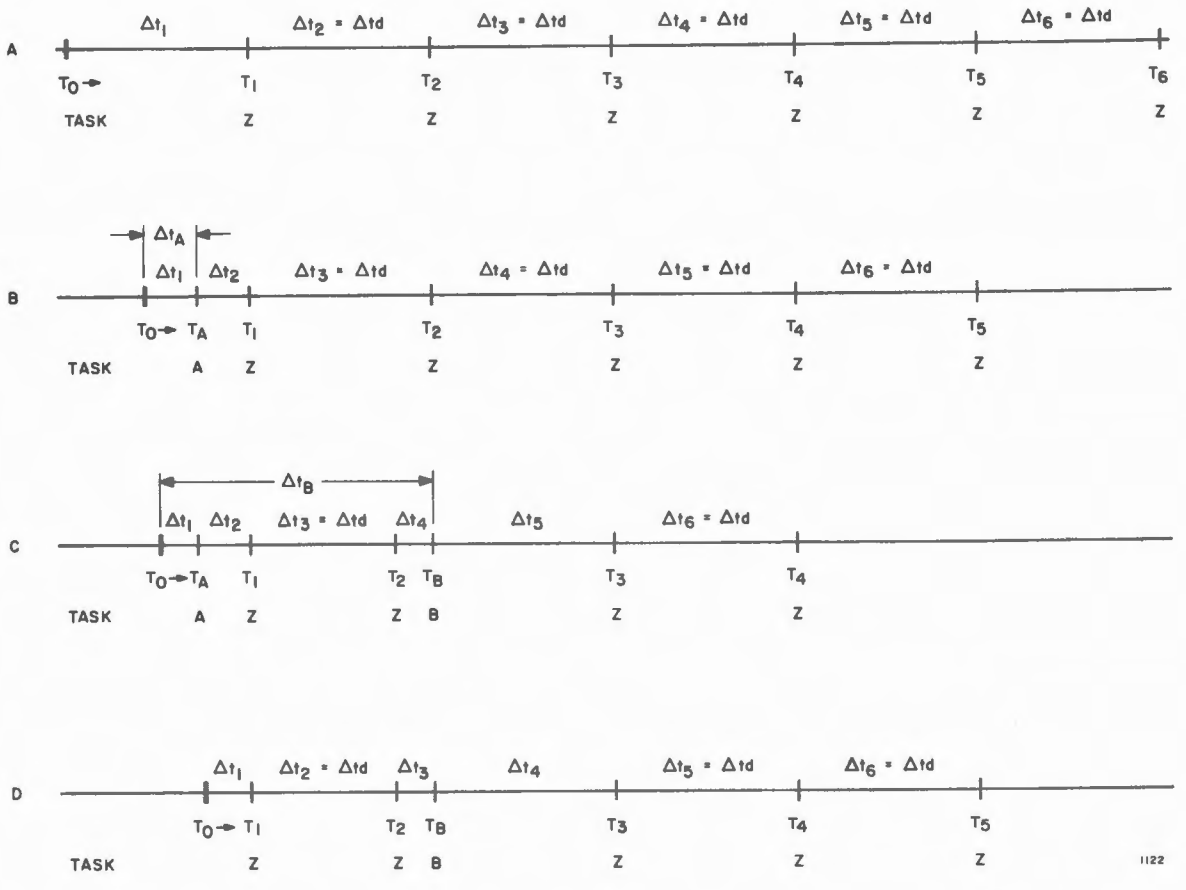

Figure 12-5. Scheduling of Tasks

C, the execution of Task B should start at time T_B. As the execution of Task B is requested, routine WAITLIST computes the new Δt's (Δt_4 and Δt_5) which are placed into locations LST1 +2 and LST1 +3. The former time interval Δt_5 (scale B) is moved into location LST1 + 4 and becomes the new Δt_6 (scale C). The former time interval Δt_6 (scale B) is pushed out. Similarly, the address of Task B is entered into location LST2 +2, the addresses of the Dummy Tasks in place 5 are pushed into place 6, and the Dummy Task in place 6 is pushed out.

12-81. If place 6 of the Waitinglist is occupied by a genuine Task and this Task is pushed out, an alarm is caused. If no place is available for a requested Task, an alarm is also caused.

12-82. As time passes, the interval Δt_1 decreases (T_0 increases) and the content of counter TIME3 (040000 - Δt_1), which is incremented every 10 msec, increases until the quantity Δt_1 is nullified. At the

instant Δt_1 becomes zero, counter TIME3 overflows. As counter TIME3 overflows, signal RPl is generated in the Interrupt Priority Control which in turn causes the SQ to execute instruction RPT (table 15-6). At the same time, the Priority Control supplies address 02000. Consequently RPT Transfer Routine TIME3RPT (table 2-6) is executed after instruction RPT to store the content of registers Z, B, A, and Q in E memory. Instruction TC T3RUPT of routine TIME3RPT finally transfers control to major routine T3RUPT of the Task Control. The T3RUPT routine moves all Δt's and addresses up one place in the Waitinglist, enters a Dummy Task in place 6, and initiates the execution of the Task whose address was last contained in location LST2. After the completion of that Task, control is returned to routine T3RUPT as indicated in figure 12-4. If the execution of another Task came due in the meantime (indicated by another overflow of counter TIME3), routime T3RUPT is executed once more to initiate the execution of the next Task. If the execution of another Task did not come due in the meantime, routine T3RUPT transfers control back to the Job which was interrupted.

12-83. Scale D of figure 12-5 indicates the conditions existing a few seconds after the execution of Task A has been initiated. The next task to be executed is Dummy Task Z whose address is contained in location LST2. After Dummy Task Z is executed another Dummy Task is executed whose address is contained in location LST2 + 1. The address of Task B is stored in location LST2 + 2. All other locations of the Addresslist store the address of the Dummy Task. Locations LST1, LST1 + 3, and LST1 + 4 store Δt_d, while locations LST1 + 1 and LST1 + 2 store computed time intervals. The Dummy Task is entered into the sixth place of the Waitlist every time routine T3RUPT moves all Δt's and addresses up one place.

12-84. WAITLIST ROUTINE

12-85. Figure 12-6 illustrates the functional flow of the WAITLIST routine, which may be entered at location WAITLIST only. A detailed flow chart of the WAITLIST routine is shown in attachment 12-3.

12-86. CALLING SEQUENCE

12-87. To enter a requested Task on the Waitlist, the WAITLIST routine is entered by means of the following calling sequence which is

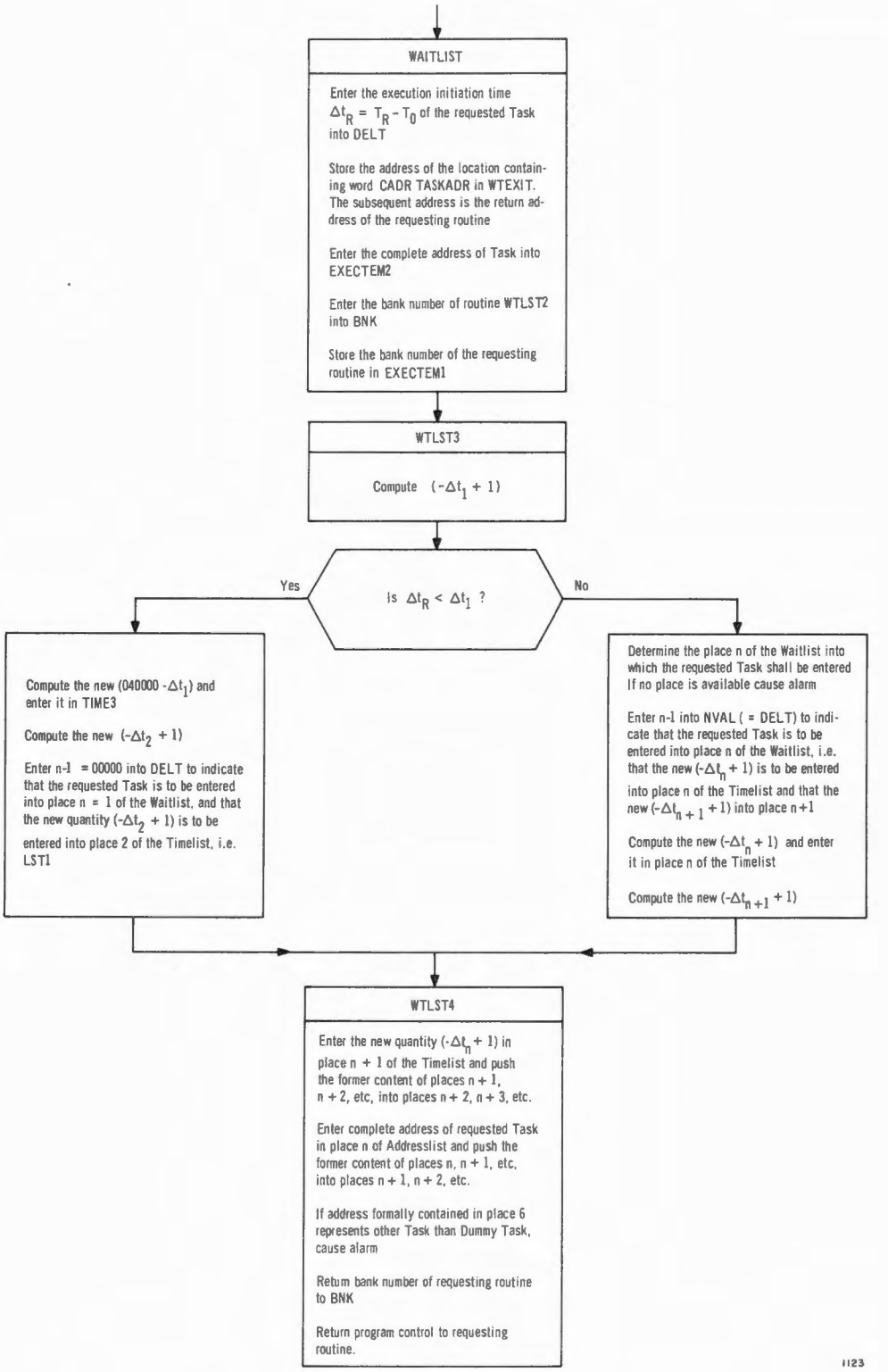

Figure 12-6. WAITLIST Routine Functional Flowchart

a part of the requesting routine:

```
CAF     DELTAT
TC      WAITLIST
CADR    TASKADR
```

Note the similarity between this calling sequence and the calling sequences of paragraphs 12-21 and 12-22. The symbol DELTAT is used as an example and represents the location within the requesting routine which contains the time interval $\Delta t_R = T_R - T_0$ (expressed in 10 msec units). This interval occurs between time T_R when the execution of the requested (genuine) Task R shall be initiated and present time T_0. The symbol TASKADR is used as an example and CADR TASKADR represents the complete address of the requested Task. The calling sequence must be preceded by an INHINT instruction, and followed by a RELINT instruction to prevent the interruption of the loading operation.

12-88. ENTRY AT WAITLIST

12-89. When the WAITLIST routine is entered at location WAITLIST (in bank 01), it first stores the time interval $\Delta t_R = T_R - T_0$ of the requested Task in register DELT, the address of word CADR TASKADR in register WTEXIT, and the complete address of the requested Task in register EXECTEM2. Then the bank code 04 is entered into register BNK to switch to bank 04 where minor routine WTLIST2 is located. The bank number of the requesting routine is stored in register EXECTEM1.

12-90. Minor routine WTLST2 compares the intended Task execution initiation time in place 1 of the Waitinglist with the intended Task execution initiation time being entered. This is accomplished by transferring the quantity (040000 - Δt_1) from counter TIME3 to the CP and calculating the quantity (Δt_1 + 1). (See attachment 12-3 and paragraph 12-95 for details of this operation.) Thereafter, the value Δt_R of the requested Task stored in register DELT is compared against the value Δt_1.

12-91. If $\Delta t_R < \Delta t_1$, the Δt_R becomes the new Δt_1 and the new quantity (040000 - Δt_1) is computed and entered into counter TIME3. Then the new Δt_2 and the quantity ($-\Delta t_2$ + 1) are calculated. The quantity 00000 is entered into register DELT to indicate that the requested Task is to be entered in place 1 of the Waitinglist and quantity ($-\Delta t_2$ + 1) is to entered into the second place of the Timelist, which is

LST1. (See paragraphs 12-96 and 12-97 for details.) Minor routine WTLST4 enters the new quantity $(-\Delta t_2 + 1)$ into LST1. The former content of location LST1 is moved to location LST1 + 1 and beomes the new quantity $(-\Delta t_3 + 1)$. The former content of location LST1 + 1 is moved to location LST1 + 2 and becomes the new quantity $(-\Delta t_4 + 1)$. The former content of location LST1 + 2 is moved to location LST1 + 3 and becomes the new quantity $(-\Delta t_5 + 1)$. The former content of location LST1 + 3 is moved to location LST1 + 4 and becomes the new quantity $(-\Delta t_6 + 1)$. The former content of location LST1 + 4 is pushed out and discarded.

12-92. If $\Delta t_R > \Delta t_1$, the Task presently occupying the first position of the Waitinglist remains there; the requested Task has to be entered into one of the other five places of the Waitinglist. To find the proper place, Δt_R is compared against $(\Delta t_1 + \Delta t_2)$, $(\Delta t_1 + \Delta t_2 + \Delta t_3)$, through $(\Delta t_1 + \Delta t_2 + \Delta t_3 + \Delta t_4 + \Delta t_5 + \Delta t_6)$ until a value is found that is larger than Δt_R. If $\Delta t_R < (\Delta t_1 + \Delta t_2)$, the quantity 00001 is entered into register NVAL (same as DELT) indicating the requested Task has to be entered into the second place of the Waitinglist. If $\Delta t_R < (\Delta t_1 + \Delta t_2 + \ldots\ldots + \Delta t_n)$, the quantity (n-1) is entered into register NVAL indicating the requested Task has to be entered into the n^{th} place of the Waitinglist. If $\Delta t_R > (\Delta t_1 + \Delta t_2 + \Delta t_3 + \Delta t_4 + \Delta t_5 + \Delta t_6)$, an alarm is caused. This condition can only occur if six genuine Tasks are occupying the Waitinglist, or five genuine Tasks and one Dummy Task. Once it has been determined which place in the Waitinglist has to be occupied by the requested Task, the new quantities $(-\Delta t_n + 1)$ and $(-\Delta t_{n+1} + 1)$ are computed. The quantity (n-1) is kept in register Q indicating the quantity $(-\Delta t_n + 1)$ is to be entered into place n and the quantity $(-\Delta t_{n+1} + 1)$ is to be entered into place n+1. The quantity $(-\Delta t_n + 1)$ is entered into place n of the Timelist. (See paragraphs 12-96, 12-98, and 12-99 for details.) Minor routine WTLST4 enters the quantity $(-\Delta t_{n+1})$ into place n+1 of the Timelist and moves the former content of places n+1, n+2, n+3, etc., up one place.

12-93. After rearranging the Timelist, minor routine WTLST4 enters the complete address of the requested Task into Addresslist place n and pushes the Task addresses formerly located in Addresslist places n, N+1, n+2, etc., into places n+1, n+2, n+3, etc. The address formerly contained in the last place of the Addresslist is tested. If this address is not the address of the Dummy Task, an alarm is caused. If the address is that of the Dummy Task, the bank number of the requesting routine is returned to register BNK and program control is returned to the requesting routine.

12-94. DETAILED ANALYSIS OF ROUTINES WTLST2 THROUGH WTLST5

12-95. The computation of the new quantity $(-\Delta t_1 + 1)$ (paragraph 12-90) depends on whether counter TIME3 has or has not overflowed since instruction INHINT (paragraph 12-87) was executed on entering the WAITLIST routine. Refer to attachment 12-3. Assume first that counter TIME3 contains quantity 30000 or $(040000 - \Delta t_1)$, in which case Δt_1 = 10000 (40.96 sec). The quantity 30000 is complemented, entered into register A, and increased by quantity 10000. Thus the content of A, c(A), becomes 157777. Since c(A) < -0, the first CCS A instruction changes c(A) to 017777. Thereafter, the quantity 150001 (50001 read out of F memory) is added to 017777 and 170000 remains in A. The quantity 170000 or -07777 represents the quantity $(-\Delta t_1 + 1)$. In addition assume counter TIME3 contains 37777 in which case Δt_1 = 1 (10 msec). The quantity 37777 is complemented, entered into A, and increased by 10000. Thus, c(A) = 150000. Since c(A) < - 0, instruction CCS A changes c(A) to 027776. Thereafter, the quantity 150001 is added to 027777 and 177777 (-0) remains in A. Next, assume counter TIME3 contains 00000 indicating overflow occurred within the last 10 msec in which case $\Delta t_1 = 0$. The quantity 00000 is complemented, entered into A, and increased by 10000. Thus, c(A) = 010000. Since c(A) > + 0, instruction CCS A changes c(A) to 007777; the quantity 140001 is added to it; the sum is complemented and becomes 027777. Thereafter, the quantity 150001 is added to 027777 and 000001 remains in A. Finally, assume counter TIME3 contains 00001 indicating overflow occurred more than 10 msec ago but less than 20 msec ago (Δt = -1). In this case, the final quantity remaining in A is 000002. These four examples prove the content of A always represents the quantity $(-\Delta t_1 + 1)$ after the execution of instruction AD OCT 50001.

12-96. Executing instruction AD DELT after instruction AD OCT50001 enters the quantity $-(\Delta t_1 - \Delta t_R) + 1$ into register A. Whenever $\Delta t_R < \Delta t_1$, the content of A is -0(177777) or less than -0. In this case, the operations described in paragraph 12-97 occur. Whenever $\Delta t_R \geq \Delta t_1$, the content of A is +1 (000001) or larger and the operation described in paragraphs 12-98 and 12-99 occurs.

12-97. If the operation of the second CCS A instruction indicates $\Delta t_R < \Delta t_1$, the value $-\Delta t_R$ is entered into A and the quantity 20000 is added twice. Then the result, $(040000 - \Delta t_R)$, is transferred to register TIME3, and becomes the new quantity $(040000 - \Delta t_1)$. Simultaneously the former content of register TIME3, the old quantity $(040000 - \Delta t_1)$, is transferred into A. (The former content of TIME3

during the execution of instruction CS TIME3. This instruction is the first instruction of routine WTLST2 providing counter TIME3 has been incremented since the transfer.) Next, the quantities 140000 and Δt_R are added to the old quantity (040000 - Δt_1). The sum (040000 - Δt_1) $_{old}$ + 140000 + Δt_R = -($\Delta t_{1\ old}$ - Δt_R) + 1 is transferred to location DELT. After the execution of instructions CAF OCT 00000 and XCH DELT, register DELT contains 00000 and register A again contains -($\Delta t_{1\ old}$ - Δt_R) + 1 which is the new quantity (-Δt_2 + 1).

12-98. In case the operation of the second CCS A instruction indicates that $\Delta t_R \geq \Delta t_1$, register A contains -($\Delta t_1$ - Δt_R) after the execution of instruction CCS A. Then the old quantity (-Δt_2 + 1) contained in register LST1 is added to the value -(Δt_1 - Δt_R) and the value -(Δt_1 + Δt_2 - Δt_R) + 1 remains in A. The first CCS A instruction of subroutine WTLST5 tests this value contained in A to determine whether the requested Task should occupy the second place in the Waitlist or a later one. If c(A) $\leq$ -0, (Δt_1 + Δt_2) > Δt_R, the requested Task must be entered into the second place in the Waitlist. In this case, the value (Δt_1 + Δt_2 - Δt_R) - 1 is entered into A, program control is transferred to subroutine WTLST2, and the address of quantity 00001 is kept in register Q. If c(A) > +0, the value -(Δt_1 + Δt_2 - Δt_R) is kept in A and the content (-Δt_3 + 1) of register LST1 + 1 is added to it, resulting in -(Δt_1 + Δt_2 + Δt_3 -Δt_R) + 1. The second CCS A instruction tests this value contained in A to determine whether the requested Task should occupy the third place in the Waitlist or a later one. If c(A) < 177777, the value (Δt_1 + Δt_2 + Δt_3 - Δt_R) -1 is entered into A, program control is transferred to routine WTLST2 and the address of quantity 00002 is kept in Q. If c(A) > +0, the content of register LST1 + 2 is added, etc. These tests are continued until a test indicates that c(A) < -0, in which case the value (Δt_1 + Δt_2 + Δt_3 + Δt_4 - Δt_R) -1, or (Δt_1 + Δt_2 + Δt_3 + Δt_4 + Δt_5 - Δt_R) -1, or (Δt_1 + Δt_2 + Δt_3 + Δt_4 + Δt_5 + Δt_6 - Δt_R) -1 is entered into A. Program control is transferred to routine WTLST2 and the address of quantity 00003, 00004, or 00005 is kept in register Q. In the event that c(A) > +0, when the fifth CCS A instruction is executed, an alarm is caused.

12-99. Upon entering subroutine WTLST2, the value (Δt_1 + Δt_2 - Δt_R) - 1, (Δt_1 + Δt_2 + Δt_3 - Δt_R) - 1, or (Δt_1 + Δt_2 + Δt_3 + Δt_4 + Δt_5 + Δt_6 - Δt_R) - 1 is transferred to register Q. The quantity 00001, 00002, or 00005 is entered into register A and then transferred to register NVAL. Next, the new Δt's are calculated. Assume (Δt_1 + Δt_2 + Δt_3) < Δt_R < (Δt_1 + Δt_2 + Δt_3 + Δt_4). In this case, Δt_1 through Δt_3 remain unchanged, Δt_R -(Δt_1 + Δt_2 + Δt_3) becomes the new Δt_4 and (Δt_1 + Δt_2 + Δt_3 + Δt_4) - Δt_R becomes the new Δt_5. The new quantity (-Δt_4 + 1) is entered into register LST1 + 2 and the new quantity (-Δt_5 + 1) is entered into the accumulator A.

12-100. T3RUPT ROUTINE

12-101. Figure 12-7 illustrates the functional flow of routine T3RUPT. The principle entrance to this routine is at location T3RUPT and the routine is re-entered at location TASKOVER after the execution of a Task. Minor routine NBRESUME, which concludes major routine T3RUPT, can also be used by other major routines in which case T3RUPT is entered at NBRESUME. A detailed flow chart of routine T3RUPT is shown in attachment 12-3.

12-102. INITIATING ACTIONS AND ENTRY AT T3RUPT

12-103. When counter TIME3 overflows, the execution of the most imminent Task (Task in place 1) is initiated by the actions described in paragraph 12-77. When routine T3RUPT is entered, registers ZRUPT, BRUPT, ARUPT, and QRUPT contain the information which was last contained in registers Z, B, A, and Q during the execution of the Job interrupted by the overflow of counter TIME3. Routine T3RUPT first stores the bank number of the interrupted Job in register BANKRUPT and the overflow bit (ov) associated with the interrupted Job in register OVRUPT. At this point all information pertaining to the interrupted Job has been saved.

12-104. When minor routine T3RUPT2 (in bank 01) is entered, counter TIME3 may contain one of the following quantities:

a. The counter contains 00000 if an overflow, which initiated the execution of T3RUPT, occurred during normal operation (paragraph 12-103) and no counter incrementation occurred since then.

b. The counter contains 00001 if an overflow, which initiated the execution of T3RUPT, occurred during normal operation and one additional counter incrementation occurred since then.

c. The counter contains 37777 if an overflow, which initiated the execution of T3RUPT, occurred between the instant an instruction INHINT of program section Fresh Start and Restart was executed and the instant the quantity 37777 was entered into counter TIME3. (Since instruction RELINT of program section Fresh Start and Restart is executed later, major routine T3RUPT cannot be executed

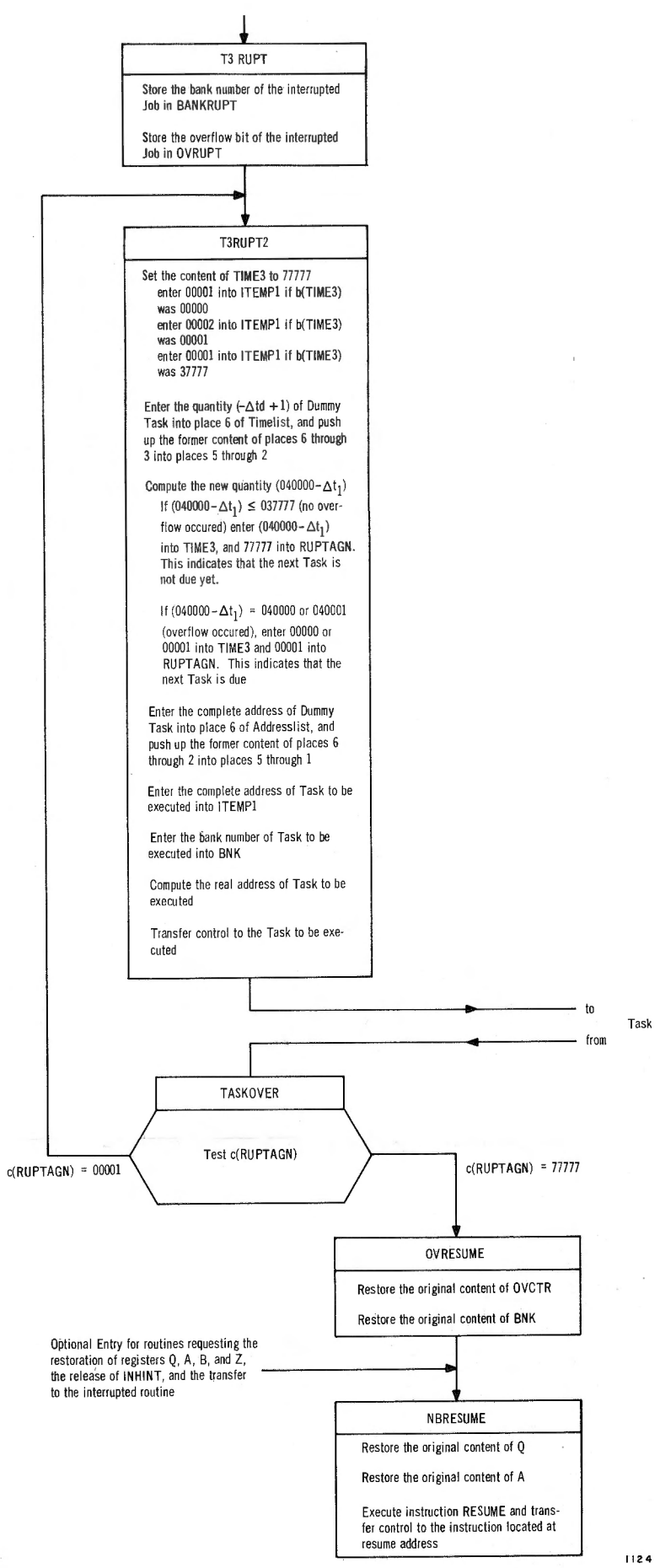

Figure 12-7 T3RUPT Routine Functional Flowchart

immediately after the overflow of TIME3. Counter TIME3 will contain 37777 when the execution of T3RUPT is started.) Rarely does counter TIME3 contain 37777 during a T3RUPT.

12-105. Minor routine T3RUPT2 clears counter TIME3 by setting it to 77777; entering 00001 into register ITEMP1 if the former content of TIME3 was 00000 or 37777; entering 00002 into register ITEMP1 if the former content of TIME3 was 00001. Next, the quantity $(-\Delta t_d + 1) = 57777_8 - 81.92$ sec of the Dummy Task is entered into place 6 of the Timelist and the former contents of places 6 through 3 are pushed into places 5 through 2. Thereafter, the quantity $(040000 - \Delta t_1)$ for the Task moving from place 2 into place 1 is computed by adding the following:

a. The old quantity $(-\Delta t_2 + 1)$ previously contained in the second place of the Timelist.

b. The content of ITEMP1

c. The quantity 17777 twice

d. The content of counter TIME3.

Counter TIME3 contains 77777 if it has not been incremented since routine T3RUPT3 was entered or 00001 if it was incremented once. If no overflow occurs during the addition, the new quantity $(040000 - \Delta t_1)$ is entered into counter TIME3 and 77777 is entered into RUPTAGN indicating the execution of the next Task is not due yet. If overflow occurs during the addition, the new quantity $(040000 - \Delta t_1)$ can be only 040000 or 040001, and 00000 or 00001 is entered into counter TIME3. The overflow bit 00001 is entered into register RUPTAGN indicating the execution of the next Task is due.

12-106. After rearrangement of the Timelist (paragraph 12-103), the complete address of the Dummy Task is entered into place 6 of the Addresslist and the contents of places 6 through 2 are pushed into places 5 through 1. The complete address of the Task to be executed now is moved from place 1 into register ITEMP1. The bank number of the same Task is entered into register BNK. Finally, the real address of the requested Task is computed and program control is transferred to that Task.

12-107. Every Task is terminated by instruction TC TASKOVER which returns program control to minor routine TASKOVER of routine T3RUPT, register RUPTAGN is tested. If register RUPTAGN contains 00001, program control is transferred to minor routine T3RUPT2

and the execution of the next Task is initiated. If register RUPTAGN contains 77777, program control is transferred to minor routines OVRESUME and NBRESUME which restore the content of registers OVCTR, BNK, Q, and A for the execution of the interrupted Job. Instruction RESUME (last instruction of the T3RUPT routine) restores the content of registers B and Z and causes the computer to resume the execution of the interrupted Job. After the execution of instruction RESUME, the first instruction of the interrupted Job is contained in register B. Instruction RESUME also releases the inhibition of program interruptions set by the RPT instruction. The RPT instruction was executed when counter TIME3 overflowed and it brought routine T3RUPT into action.