

cultura nello sviluppo

S E R V I Z I D I Programmazione

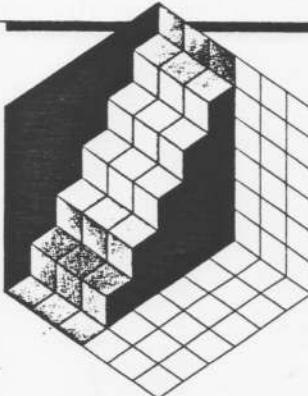
LC262

FORMAZIONE E

Testo di studio

ADDESTRAMENTO

CICS - COBOL



IBM

cultura nello sviluppo

S E R V I Z I D I Programmazione

LC262

FORMAZIONE E in ambiente

Testo di studio

ADDESTRAMENTO CICS - COBOL

Proprio con AVMED-MIS gli operatori della tua azienda saranno sempre al corrente di tutte le esigenze dei tuoi clienti e potrai così offrire un servizio sempre più completo e soddisfacente.

1984 - ADVISORIAL INFORMATION

IBM

S E R V I Z I D I
R e l a t i o n e s

L O R M A S I O N E E
In s u p p l i e s

A D D E S T R A M E N T O C I C S - C O R O F

Questa pubblicazione, edita dai Servizi di Addestramento della IBM SEMEA, ha scopi esclusivamente didattici; essa pertanto non sostituisce in alcun modo le pubblicazioni tecniche relative agli argomenti trattati.
La IBM SEMEA, pur intendendo procedere a periodiche revisioni di questa pubblicazione, non si assume alcun impegno sul suo livello di aggiornamento.

© Copyright IBM SEMEA - 1992

PREFAZIONE

Il corso che stai per iniziare e' un corso autodidattico guidato che fa uso di un Personal Computer.

L'oggettivo autodidattico vuole rammentare che e' tua responsabilita' pianificare le attivita' da svolgere durante il corso, cioe' scegliere il ritmo di lavoro, studiare gli argomenti delle lezioni, svolgere gli esercizi proposti.

Cio' non significa pero' che sei lasciato a te stesso, anzi abbiamo predisposto le cose in modo che tu possa trovarsi nelle migliori condizioni di apprendimento: sarai per questo guidato e stimolato dal dialogo col Personal Computer.

Il Personal Computer sostituisce il docente del corso, con il vantaggio che le redini delle lezioni sono nelle tue mani, garantendoti il diritto (se cosi' si puo' dire) di interrompere, chiudere delucidazioni o ripetere un'intera lezione.

Durante lo svolgimento del corso alternerai due tipi di attivita':

1. lo studio del TESTO DEL CORSO, contenente gli argomenti del corso stesso, seguito da
2. la LEZIONE INTERATTIVA col Personal Computer, relativa agli argomenti appena studiati.

In particolare la Lezione Interattiva ti invita a rispondere su quanto hai studiato e ti da l'opportunita' di consolidare le tue conoscenze mettendo in evidenza le cose piu' importanti, illustrando varie esemplificazioni e proponendo esercizi.

Per questo il materiale del corso consiste di:

1. una INTRODUZIONE,
2. una GUIDA ALLO STUDIO,
3. un TESTO DEL CORSO,
4. una serie di LEZIONI INTERATTIVE con il Personal Computer,
5. un MANUALE DEGLI ESERCIZI con relative soluzioni,
6. un ESTRATTO DELLE DISPENSE ufficiali IBM .

L'INTRODUZIONE, la GUIDA ALLO STUDIO, il TESTO DEL CORSO ed il MANUALE DEGLI ESERCIZI costituiscono il VOLUME 1, cioe' questo stesso volume che stai leggendo ora.

Il VOLUME 2 e' costituito, invece, dagli ESTRATTI DELLE DISPENSE ufficiali IBM (in lingua inglese), alla cui consultazione sarai esplicitamente inviato quando necessario, ma che comunque ti invitiamo ad esaminare frequentemente per avere un riferimento ufficiale sull'argomento.

L'INTRODUZIONE che segue questa Prefazione ti racconta quali sono i contenuti e gli obiettivi del corso.

La GUIDA ALLO STUDIO, che segue l'Introduzione, ti indica la sequenza temporale delle attivita' da svolgere per portare a termine il corso nel tempo previsto e con il massimo profitto. Essa consiste di una serie di tabelle indicanti ciascuna il carico di lavoro da svolgere in mezza giornata.

Per tutto cio' che riguarda l'uso del MANUALE DEGLI ESERCIZI sarai guidato direttamente durante le lezioni interattive col Personal Computer.

Detto questo non ci resta altro che augurarti BUONO STUDIO !

Gli Autori.

INDICE GENERALE VOLUME 1

PARTE 1

| | |
|-------------------------|------------|
| INTRODUZIONE | Divisore 1 |
| GUIDA ALLO STUDIO | Divisore 2 |
| TESTO DEL CORSO | Divisore 3 |

PARTE 2

| | |
|------------------------------|------------|
| MANUALE DEGLI ESERCIZI | Divisore 4 |
| SOLUZIONI | Divisore 5 |

1. PRIMO ESEMPIO

1. STRAN
2. STRAN
3. STRAN
4. STRAN

5. STRAN
6. STRAN

INTRODUZIONE

Il corso di PROGRAMMAZIONE CICS/VS si rivolge a programmatore di applicazioni che abbiano esperienza di programmazione COBOL in ambiente "batch" che utilizzi Data Set VSAM, Data Base DL/I o DB2.

Per comodita' di studio, il corso e' stato suddiviso in nove sessioni, ciascuna della durata di mezza giornata, per un totale di quattro giorni e mezzo.

Al termine del corso lo studente conoscerà la maggior parte dei COMANDI CICS/VS e sarà in grado di scrivere applicazioni interattive in ambiente CICS-DLI e/o CICS-VSAM e/o CICS-DB2.

Per raggiungere questo obiettivo, lo studente sarà invitato a costruire, durante lo svolgimento di dieci esercitazioni, una serie di programmi esemplificativi di alcune situazioni ricorrenti in ambiente on-line.

Per i piu' importanti programmi realizzati durante il corso e' prevista anche la simulazione dell'esecuzione del programma, per consentire una visione "concreta" dell'esito del lavoro.

Inoltre, per alcuni argomenti particolarmente critici, abbiamo fatto uso di animazioni grafiche che forniranno anche una spiegazione visiva dei concetti esposti.

Il referto fa riferimento alla programmazione in COBOL per la gestione degli ordini di fabbrica nel magazzino e nella fabbrica con l'elenco delle attività di gestione degli ordini.

Il referto si articola in due parti: la prima riguarda le attività che riguardano gli ordini di fabbrica, mentre la seconda riguarda gli ordini di magazzino.

Nella prima parte del referto vengono elencate le attività di gestione degli ordini di fabbrica, con l'elenco delle attività di gestione degli ordini di magazzino.

La seconda parte del referto riguarda gli ordini di fabbrica, con l'elenco delle attività di gestione degli ordini di fabbrica.

Il referto si articola in due parti: la prima riguarda gli ordini di fabbrica, mentre la seconda riguarda gli ordini di magazzino.

Questa Guida allo Studio e' suddivisa in nove SESSIONI di mezza giornata, ognuna indicante la sequenza temporale delle attivita' da svolgersi con i relativi obiettivi.

Come abbiamo anticipato, le attivita' sono di due tipi: studio del Testo del Corso e lezione interattiva col Personal Computer.

In questa Guida allo Studio la prima attivita' e' indicata come TESTO, mentre la seconda e' indicata col termine CBT , che e' l'abbreviazione di Computer Based Training.

Percio', nel leggere la guida ad una SESSIONE, dove c'e' l'indicazione TESTO, si intende invitare l'allievo a studiare il Capitolo indicato del TESTO DEL CORSO; dove c'e' l'indicazione CBT, si intende invitare l'allievo a seguire la lezione interattiva col Personal Computer, il cui numero e' indicato vicino.

Per esempio la prima attivita' della SESSIONE 1 e' lo studio del Capitolo 1 sul TESTO DEL CORSO; la terza attivita' e' la lezione interattiva del Capitolo 1 e cosi via.

Alcune sessioni (in particolare la 3 e la 4) potrebbe durare anche piu' di mezza giornata. Non sara' difficile recuperare l'eventuale ritardo nelle sessioni successive che sono piu' leggere.

...di recupero - Recupero di ...

...di recupero -

...di recupero -

...di recupero -

...di recupero -

SESSIONE 1

XX METODO DI STUDIO E ARGOMENTI DELLA LEZIONE XX

TESTO Capitolo 1:

- La documentazione di applicazioni interattive.

CBT Capitolo 1:

- Documentazione.
- Presentazione della applicazione di Inquiry Update.

TESTO Capitolo 2:

- Il Terminale 3270.

CBT Capitolo 2:

- Il Terminale 3270.
- Principi di funzionamento.
- Animazione.

XX OBIETTIVI DELLA SESSIONE 1 XX

- Interpretare la documentazione di una applicazione interattiva.
- Il terminale 3270 : descriverne i principi di funzionamento.
- Esercizio : Lettura della documentazione della applicazione di Inquiry Update.

SESSIONE 2

METODO DI STUDIO E ARGOMENTI DELLA LEZIONE

TESTO Capitolo 3:

- Le macro del B.M.S.

CBT Capitolo 3:

- Le mappe del B.M.S.
- Macro per la definizione di mappe.
- Esercizi 1 e 2.

TESTO Capitolo 4:

- L'ambiente di programmazione CICS/VS.

CBT Capitolo 4:

- Ambiente di programmazione COBOL-CICS.

OBIETTIVI DELLA SESSIONE 2

- Definire le mappe del BMS mediante macroistruzioni.
- Esercizio 1: Costruzione delle macroistruzioni per la definizione di mappe dell'applicazione I/U.
- Inserire comandi CICS nel linguaggio di programmazione ospite.
- Descrivere la sintassi dei comandi CICS.

XX
METODO DI STUDIO E ARGOMENTI DELLA LEZIONE
XX

TESTO Capitolo 5:

- Comandi BMS e TC.

CBT Capitolo 5:

- Programmazione con comandi BMS.
- Dichiarazione delle aree.
- Invio e ricezione di mappe.
- Gestione delle condizioni eccezionali.
- Esercizio 2: Realizzazione del programma di gestione del Menù.
- Simulazione del programma.

XX
OBETTIVI DELLA SESSIONE 3
XX

- Codificare i comandi CICS per l'invio e la ricezione di mappe.
- Stendere un programma completo in ambiente COBOL/CICS.

SESSIONE 6

METODO DI STUDIO E ARGOMENTI DELLA LEZIONE

- | | |
|-----------|---|
| TESTO | Capitolo 6: - Il Program Control (escluso paragrafo 6.10) |
| CBT | Capitolo 6: - Elementi di Transazione CICS. |
| (*1)TESTO | Capitolo 7: - I Data Base DL/I. |
| (*1)CBT | Capitolo 7: - Accesso a Data Base DL/I. - Esercizio 3: costruzione del programma di I/U. - Simulazione del programma di I/U. |
| (*2)TESTO | Capitolo 8: - I Data Set VSAM. |
| (*2)CBT | Capitolo 8: - Accesso a file VSAM. - Esercizio 4: costruzione del programma di I/U. - Simulazione del programma di I/U. |
| (*3)TESTO | Capitolo 9: - I Data Base DB2. |
| (*3)CBT | Capitolo 9: - Accesso a Data Base DB2. - Esercizio 5: costruzione del programma di I/U. - Simulazione del programma di I/U. |

OBIETTIVI DELLA SESSIONE 4

- Passare dati e controllo tra i programmi in ambiente CICS.
- Codificare comandi per l'accesso a Data Base DLI in ambiente CICS
- Codificare comandi per l'accesso a files VSAM in ambiente CICS

- Codificare comandi per l'accesso a Data Base DB2 in ambiente CICS.
- Esercizio 3: costruzione di un completo programma per realizzare l'applicazione di Inquiry/Update in ambiente CICS/DLI.
- Esercizio 4: costruzione di un completo programma per realizzare l'applicazione di Inquiry-Update.
- Simulazione dell'esecuzione del programma di Inquiry-Update

N.B.: gli argomenti contrassegnati con (*1) riguardano il DLI, quelli contrassegnati con (*2) riguardano il VSAM mentre quelli contrassegnati con (*3) riguardano il DB2.

SESSIONE 5

XX METODO DI STUDIO E ARGOMENTI DELLA LEZIONE XX

TESTO Capitolo 6:

- Il Program Control (solo paragrafo 6.10).

CBT Capitolo 10:

- L'organizzazione di una transazione pseudoconversazionale.
- Esercizio 6: Versione pseudoconversazionale del programma di Menu'. Animazione.

TESTO Capitolo 10:

- Transient Data.

CBT Capitolo 11:

- Il transient Data
- Esercizio 7: Applicazione del TD per un programma di Order Entry. Animazione.

XX OBIETTIVI DELLA SESSIONE 5 XX

- Spiegare in cosa consiste la tecnica pseudoconversazionale di gestione del dialogo.
- Esame di un programma realizzato con tecnica pseudoconversazionale.
- Come gestire le code di Transient Data.
- Realizzazione di una applicazione di order entry e stampa mediante code di Transient Data.

SESSIONE 6

XX METODO DI STUDIO E ARGOMENTI DELLA LEZIONE XX

TESTO Capitolo 11:

- Il Temporary Storage.

CBT Capitolo 12:

- Il Temporary Storage.
- Esercizio 8: Applicazione del TS per il programma di Order Entry.
- Animazione.

TESTO Capitolo 12:

- L'Interval Control.

CBT Capitolo 13:

- I servizi dell'Interval Control.
- Partenza a tempo dei task CICS.

XX OBIETTIVI DELLA SESSIONE 6 XX

- Come gestire una coda di Temporary Storage.
- Realizzazione del programma di order entry e stampa mediante coda di TS e TD.
- Come richiedere all'Interval Control Program la partenza di un task.
- Realizzazione del programma di order entry e stampa mediante task a tempo.

SESSIONE 7 E SESSIONE 8

XX
METODO DI STUDIO E ARGOMENTI DELLA LEZIONE
XX

TESTO Capitolo 13:

- Il Paging

N CBT Capitolo 14:

- La tecnica di Paging.
- Esercizio 9: Costruzione di un programma che richiede i servizi di Paging con l'uso del VSAM
- Simulazione dell'esecuzione programma.

XX
OBIETTIVI DELLE SESSIONI 7 E 8
XX

- I servizi del PAGING : come richiederli attraverso comandi.
- Realizzazione di un programma che richiede la paginazione.
- Simulazione dell'esecuzione del programma.

SESSIONE 9

METODO DI STUDIO E ARGOMENTI DELLA LEZIONE

- TESTO Capitolo 14:
 - Indirizzamento di aree esterne.
- TESTO Capitolo 15:
 - Comandi per la gestione delle risorse.
- TESTO Capitolo 16:
 - EXECUTION DIAGNOSTIC FACILITY.

OBIETTIVI DELLA SESSIONE 9

- conoscere le possibilità di indirizzamento di aree esterne.
- utilizzare il programma interattivo di diagnostica chiamato EDF.
- conoscere la possibilità di utilizzare comandi per l'interrogazione e la modifica dello stato delle risorse.

... ATTIVITÀ DI CAGLIARI E DI MESSINA AL 1
240 GENNAIO 1942

ATTIVITÀ DI Cagliari 1.1

ATTIVITÀ DI SICILIA 1.2

ATTIVITÀ DI TRAPANI 1.3

ATTIVITÀ DI VENEZIA 1.4

ATTIVITÀ VENEZIA 1.5

ATTIVITÀ VENEZIA 1.6

ATTIVITÀ VENEZIA 1.7

ATTIVITÀ VENEZIA 1.8

ATTIVITÀ VENEZIA 1.9

ATTIVITÀ VENEZIA 1.10

ATTIVITÀ VENEZIA 1.11

ATTIVITÀ VENEZIA 1.12

ATTIVITÀ VENEZIA 1.13

ATTIVITÀ VENEZIA 1.14

ATTIVITÀ VENEZIA 1.15

ATTIVITÀ VENEZIA 1.16

ATTIVITÀ VENEZIA 1.17

ATTIVITÀ VENEZIA 1.18

ATTIVITÀ VENEZIA 1.19

ATTIVITÀ VENEZIA 1.20

ATTIVITÀ VENEZIA 1.21

ATTIVITÀ VENEZIA 1.22

ATTIVITÀ VENEZIA 1.23

ATTIVITÀ VENEZIA 1.24

ATTIVITÀ VENEZIA 1.25

ATTIVITÀ VENEZIA 1.26

ATTIVITÀ VENEZIA 1.27

ATTIVITÀ VENEZIA 1.28

ATTIVITÀ VENEZIA 1.29

ATTIVITÀ VENEZIA 1.30

ATTIVITÀ VENEZIA 1.31

ATTIVITÀ VENEZIA 1.32

ATTIVITÀ VENEZIA 1.33

ATTIVITÀ VENEZIA 1.34

ATTIVITÀ VENEZIA 1.35

ATTIVITÀ VENEZIA 1.36

ATTIVITÀ VENEZIA 1.37

ATTIVITÀ VENEZIA 1.38

ATTIVITÀ VENEZIA 1.39

ATTIVITÀ VENEZIA 1.40

ATTIVITÀ VENEZIA 1.41

ATTIVITÀ VENEZIA 1.42

ATTIVITÀ VENEZIA 1.43

ATTIVITÀ VENEZIA 1.44

ATTIVITÀ VENEZIA 1.45

ATTIVITÀ VENEZIA 1.46

ATTIVITÀ VENEZIA 1.47

2013 BROGLIARDO 10 ATTIVITÀ

2013 BROGLIARDO 11

2013 BROGLIARDO 12

2013 BROGLIARDO 13

INDICE

| | |
|--|----|
| 1 LA DOCUMENTAZIONE DI UN'APPLICAZIONE INTERATTIVA | 1 |
| 1.1 INTRODUZIONE | 1 |
| 1.2 GENERALITÀ | 1 |
| 1.3 DEFINIZIONE DI TASK UTENTE | 1 |
| 1.4 IL PANNELLO | 2 |
| 1.5 IL FLUSSO DEI PANNELLI | 3 |
| 1.5.1 La sequenza semplice | 4 |
| 1.5.2 La sequenza gerarchica | 4 |
| 1.6 CONTROLLO DEL FLUSSO DEI PANNELLI | 4 |
| 1.6.1 Controllo del flusso in caso di errore | 5 |
| 1.7 LA STRUTTURA DEI DATI | 6 |
| 1.7.1 Files tradizionali | 6 |
| 1.7.2 Data Bases DL/I | 7 |
| 1.7.3 Data Base DB2 | 7 |
| 1.8 SOMMARIO | 7 |
| 1.9 FIGURE | 8 |
| 2 IL TERMINALE 3270 | 21 |
| 2.1 INTRODUZIONE | 21 |
| 2.2 IL TERMINALE IBM 3270 | 21 |
| 2.2.1 La Tastiera | 22 |
| Tasti di digitazione dati | 22 |
| Tasti di controllo immissione | 22 |
| Tasti per inviare dati all'elaboratore | 23 |
| 2.2.2 Lo schermo | 24 |
| Il carattere attributo | 26 |
| 2.3 SOMMARIO | 28 |
| 2.4 FIGURE | 29 |
| 3 LE MACRO DEL BMS | 31 |
| 3.1 INTRODUZIONE | 31 |
| 3.2 I VANTAGGI DEL BMS | 31 |
| 3.2.1 Device Independence | 31 |
| 3.2.2 Format Independence | 32 |
| 3.3 LE MAPPE FISICHE | 32 |
| 3.4 LE MAPPE LOGICHE (o SIMBOLICHE) | 33 |
| 3.4.1 Il Formato FIELD | 34 |
| 3.4.2 Il Formato BLOCK | 35 |
| 3.4.3 Il Formato TEXT | 36 |
| 3.5 RICHIAMI | 37 |
| 3.6 MACRO PER GENERARE MAPPE | 38 |
| 3.6.1 Definizione di un MAPSET (macro DFHMSD) | 40 |
| 3.6.2 Definizione di una Mappa (macro DFHMDI) | 44 |
| 3.6.3 Definizione dei Campi (macro DFHMDF) | 45 |
| 3.7 PREPARAZIONE DELLE MAPPE LOGICHE | 48 |
| 3.8 SOMMARIO | 50 |
| 3.9 FIGURE | 51 |
| 4 L'AMBIENTE DI PROGRAMMAZIONE CICS | 55 |
| 4.1 INTRODUZIONE | 55 |
| 4.2 LA TRANSAZIONE CICS | 55 |
| 4.3 I SERVIZI DEL CICS | 55 |

| | |
|---|-----|
| 4.4 L'INTERFACCIA CICS/VIS - COMANDI | 56 |
| 4.4.1 IL FORMATO DEI COMANDI | 56 |
| 4.4.2 I VALORI DEGLI ARGOMENTI | 57 |
| 4.4.3 IL TRADUTTORE DEI COMANDI | 59 |
| 4.4.4 LE OPZIONI PER IL TRADUTTORE | 59 |
| 4.4.5 RESTRIZIONI DEL COMPILATORE COBOL | 60 |
| 4.5 IL BLOCCO EIB (EXECUTE INTERFACE BLOCK) | 60 |
| 4.6 LE CONDIZIONI ANOMALE | 61 |
| 4.6.1 Comando IGNORE | 63 |
| 4.6.2 Opzione NOHANDLE | 64 |
| 4.6.3 Resp | 65 |
| 4.7 SOMMARIO | 66 |
| 5 COMANDI BMS E TC | 67 |
| 5.1 INTRODUZIONE | 67 |
| 5.2 I/O MAPPING | 67 |
| 5.3 INPUT MAPPING | 68 |
| 5.3.1 Il Comando RECEIVE MAP | 69 |
| 5.4 OUTPUT MAPPING | 71 |
| 5.4.1 Il Comando SEND MAP | 71 |
| 5.4.2 Posizionamento del Cursore | 73 |
| 5.4.3 Modifica Dinamica dell'Attributo | 74 |
| 5.5 LE CONDIZIONI ANOMALE | 75 |
| 5.6 GESTIONE DEI TASTI FUNZIONALI | 76 |
| 5.7 FUNZIONI DEL TERMINAL CONTROL | 78 |
| 5.8 COMANDI DI TC | 78 |
| 5.8.1 Invio dati ad un terminale | 79 |
| 5.8.2 Ricezione dati da un terminale | 79 |
| 5.9 NOTE SU SEND E RECEIVE | 80 |
| 5.10 SOMMARIO | 81 |
| 6 PROGRAM CONTROL | 83 |
| 6.1 INTRODUZIONE | 83 |
| 6.2 CONSIDERAZIONI APPLICATIVE | 83 |
| 6.3 IL COMANDO LINK | 84 |
| 6.4 IL COMANDO XCTL | 85 |
| 6.5 IL COMANDO RETURN | 85 |
| 6.6 CARICAMENTO DI UNA TABELLA | 86 |
| 6.7 LA CONDIZIONE ANOMALA PGMDERR | 86 |
| 6.8 PASSAGGIO DI DATI AD ALTRI PROGRAMMI | 87 |
| 6.9 FINE ANORMALE DI UN TASK | 90 |
| 6.10 TRANSAZIONI CONVERSATORIALI E PSEUDO | 91 |
| 6.10.1 Transazione Conversazionale | 92 |
| 6.10.2 Transazione Pseudoconversazionale | 93 |
| 6.11 SOMMARIO | 94 |
| 6.12 FIGURE | 96 |
| 7 I DATA BASE DL/I | 99 |
| 7.1 INTRODUZIONE | 99 |
| 7.2 DEFINIZIONE E VISTA DI UN DATA BASE | 99 |
| 7.3 L'AMBIENTE DL/I | 100 |
| 7.4 IL PROGRAMMA CICS-DL/I | 100 |
| 7.5 I SERVIZI DL/I IN AMBIENTE ON-LINE | 100 |
| 7.5.1 SCHEDULING | 101 |
| 7.5.2 TERMINATING | 101 |
| 7.6 I COMANDI DL/I | 101 |

| | |
|---|-----|
| 7.7 LA FUNZIONE | 102 |
| 7.8 L'INDICAZIONE DEL PCB | 103 |
| 7.9 LA SCELTA DEI SEGMENTI | 104 |
| 7.9.1 Segmenti Genitori | 104 |
| 7.9.2 Segmenti Oggetti | 104 |
| 7.9.3 I/O Area di Segmento | 104 |
| 7.9.4 Lunghezza di Segmento | 105 |
| 7.9.5 Selezione di Segmento | 105 |
| 7.10 RISPOSTE AI COMANDI DL/I | 107 |
| 7.10.1 Status Code | 107 |
| 7.10.2 Gestione degli Abend | 108 |
| 7.11 SOMMARIO | 109 |
| 8 I DATA SETS VSAM | 111 |
| 8.1 INTRODUZIONE | 111 |
| 8.2 IL FCP | 111 |
| 8.3 EXCLUSIVE CONTROL DEL VSAM | 111 |
| 8.4 DEFINIZIONE DELLE AREE DI I/O | 112 |
| 8.5 LETTURA DI UN RECORD | 112 |
| 8.6 AGGIORNAMENTO DI UN RECORD | 114 |
| 8.7 CANCELLAZIONE DI RECORD | 114 |
| 8.8 SCRITTURA DI NUOVI RECORD | 115 |
| 8.9 LE CONDIZIONI ANOMALE | 116 |
| 8.10 BROWSING | 118 |
| 8.11 ELEMENTI DI UN BROWSING | 118 |
| 8.12 COMANDI DI BROWSING | 120 |
| 8.12.1 Posizionamento | 120 |
| 8.12.2 Lettura Sequenziale | 121 |
| 8.12.3 Riposizionamento | 122 |
| 8.12.4 Chiusura | 123 |
| 8.13 ELABORAZIONE SKIP-SEQUENTIAL | 123 |
| 8.14 BROWSING MULTIPLO | 124 |
| 8.15 INDICI ALTERNATIVI VSAM | 124 |
| 8.16 CONSIDERAZIONI SULL'USO DEL BROWSING | 125 |
| 8.17 LE CONDIZIONI ANOMALE DI BROWSING | 125 |
| 8.18 SOMMARIO | 127 |
| 9 I DATA BASE DB2 | 129 |
| 9.1 INTRODUZIONE | 129 |
| 9.2 DEFINIZIONE DI UN DATA BASE DB2 | 129 |
| 9.3 STRUCTURED QUERY LANGUAGE (SQL) | 129 |
| 9.4 DEFINIZIONE DELLE AREE DI I/O | 129 |
| 9.5 UNIT OF RECOVERY | 130 |
| 9.6 GLI STÄTEMENTS SQL | 131 |
| 9.7 LOCKING DB2 | 133 |
| 9.8 RISPOSTE AGLI STATEMENTS SQL | 133 |
| 9.9 GESTIONE GENERALIZZATA DEGLI ERRORI SQL | 133 |
| 9.10 PRECOMPILAZIONE DEL PGM DB2 | 134 |
| 9.11 SOMMARIO | 134 |
| 10 TRANSIENT DATA | 135 |
| 10.1 INTRODUZIONE | 135 |
| 10.2 GENERALITÀ | 135 |
| 10.3 DESTINAZIONI EXTRAPARTITION | 135 |
| 10.4 DESTINAZIONI INTRAPARTITION | 136 |

| | | |
|---------|---|-----|
| 10.5 | DESTINAZIONI INDIRETTE | 137 |
| 10.6 | SCRITTURA SU TD | 138 |
| 10.7 | LETTURA DA TD | 138 |
| 10.8 | CANCELLAZIONE DI CODE TD | 139 |
| 10.9 | PARTENZA AUTOMATICA DI TASK | 139 |
| 10.10 | LE CONDIZIONI ANOMALE DI TD | 141 |
| 10.11 | SOMMARIO | 141 |
| 11 | TEMPORARY STORAGE | 143 |
| 11.1 | INTRODUZIONE | 143 |
| 11.2 | GENERALITÀ | 143 |
| 11.3 | SCRITTURA SU TS | 144 |
| 11.4 | LETTURA DA TS | 145 |
| 11.5 | CANCELLAZIONE DI CODE TS | 146 |
| 11.6 | LE CONDIZIONI ANOMALE DI TS | 146 |
| 11.7 | SOMMARIO | 147 |
| 12 | INTERVAL CONTROL | 149 |
| 12.1 | INTRODUZIONE | 149 |
| 12.2 | ORA E DATA DI UN TASK | 149 |
| 12.3 | PARTENZA A TEMPO DI UN TASK | 149 |
| 12.4 | LETTURA DEI DATI PASSATI AD UN TASK A TEMPO | 151 |
| 12.5 | INTERRUZIONE DI UN TASK | 152 |
| 12.6 | CANCELLAZIONE DI RICHIESTE | 153 |
| 12.7 | LE CONDIZIONI ANOMALE DI IC | 153 |
| 12.8 | ALTRÉ POSSIBILITÀ | 154 |
| 12.9 | SOMMARIO | 155 |
| 13 | PAGING | 157 |
| 13.1 | INTRODUZIONE | 157 |
| 13.2 | ORA E DATA DI UN TASK | 157 |
| 13.3 | STRUTTURA DELLE PAGINE | 158 |
| 13.4 | USO DELLA DEVICE INDEPENDENCE | 158 |
| 13.5 | POSIZIONAMENTO DI MAPPE | 159 |
| 13.6 | L'OVERFLOW DI PAGINA | 162 |
| 13.7 | COSTRUZIONE DI MESSAGGI LOGICI | 163 |
| 13.8 | GESTIONE DELL'OVERFLOW | 165 |
| 13.9 | CHIUSURA DI UN MESSAGGIO LOGICO | 166 |
| 13.10 | LA TRANSAZIONE DI PAGINAZIONE | 168 |
| 13.11 | COMANDI DI PAGINAZIONE | 169 |
| 13.12 | CANCELLAZIONE DI UN MESSAGGIO LOGICO | 170 |
| 13.13 | COSTRUZIONE DI TESTI | 170 |
| 13.14 | IL COMANDO SEND TEXT | 172 |
| 13.15 | CHIUSURA DI UN TESTO | 174 |
| 13.16 | IL ROUTING | 175 |
| 13.16.1 | Gestione dell'Overflow di pagina | 178 |
| 13.17 | SOMMARIO | 179 |
| 13.18 | FIGURE | 181 |
| 14 | INDIRIZZAMENTO DI AREE ESTERNE | |
| 14.1 | INTRODUZIONE | 183 |
| 14.2 | AREE DI MEMORIA ESTERNE | 183 |
| 14.3 | ACCESSO AI BLOCCHI DI CONTROLLO CICS/VIS | 184 |
| 14.4 | ACCESSO ALLE AREE DI INPUT | 185 |
| 14.5 | SOMMARIO | 186 |

| | |
|--|-----|
| 15 COMANDI PER LA GESTIONE DELLE RISORSE | 187 |
| 15.1 INTRODUZIONE | 187 |
| 15.2 COMANDI DI INTERROGAZIONE | 188 |
| 15.3 COMANDI DI IMPOSTAZIONE | 189 |
| 16 EXECUTION DIAGNOSTIC FACILITY | 191 |
| 16.1 INTRODUZIONE | 191 |
| 16.2 COME INIZIARE UNA SESSIONE DI DEBUGGING | 191 |
| 16.3 POSSIBILITÀ DI MODIFICA | 192 |
| 16.4 ESEMPIO DI USO DELL'EDF | 193 |

1 LA DOCUMENTAZIONE DI UN'APPLICAZIONE INTERATTIVA

1.1 INTRODUZIONE

Questo capitolo iniziale ha lo scopo di informare il programmatore sulle diverse attivita' necessarie per realizzare un'applicazione TP e, in particolare, su come utilizzare la documentazione che gli verrà fornita all'inizio del suo lavoro.

1.2 GENERALITA'

Realizzare una qualsiasi applicazione, che faccia uso di un elaboratore, comporta sempre una serie di attivita', più o meno articolate, dipendenti dalla complessità dell'applicazione stessa.

Nelle generalita' dei casi, però, possiamo suddividere un tale lavoro in tre fasi (vedi figura 1.1).

La prima fase sarà quella di disegnare l'applicazione, cioè chiedersi "cosa" dovrà fare l'applicazione stessa, e "come" dovrà essere realizzata. Questa prima fase viene eseguita dagli analisti e comporta il coinvolgimento degli utenti finali, che devono essere i fruitori della nuova applicazione.

La prima fase si conclude solo quando tutte le specifiche da realizzare per la nuova applicazione saranno completamente definite e, perciò, si può ritenere chiuso il lavoro di analisi.

Si apre così la seconda fase che è quella di "sviluppo" della applicazione, che consiste nella progettazione, codifica e prova dei programmi da parte del programmatore. Questa fase è quella che normalmente comporta il maggior dispendio di risorse (tempo e uomini), come è anche evidenziato dalla figura 1.1.

Allo sviluppo segue l'ultima fase, che è quella di "implementare" l'applicazione, ovvero di fare in modo che l'utente finale possa utilizzarla. Quest'ultimo compito è responsabilità dell'amministratore di sistema, che si farà carico di assegnare tutte le risorse hardware e software necessarie al funzionamento dell'applicazione.

1.3 DEFINIZIONE DI TASK UTENTE

Perché un programmatore applicativo possa svolgere al meglio la fase di sviluppo dell'applicazione, deve essere in condizione di interpretare correttamente la documentazione che gli è fornita dagli analisti quale risultato della prima fase.

Il primo documento messo a disposizione del programmatore e' la "DESCRIZIONE DEL TASK UTENTE" (di cui la figura 1.2 e' un esempio): prima di tentarne una interpretazione, diamo alcune definizioni.

Definiamo "TASK UTENTE" o "UNITA' DI LAVORO UTENTE" come la sequenza organizzata e definita di operazioni elementari, descrivibile come un "tutto unico", necessaria per eseguire un certo lavoro.

In particolare un task utente, e' identificato da:

Unita' di azione cioè: regole fisse e conosciute per trasformare le informazioni;

Unita' di tempo cioè: esecuzione senza interruzione e in modo ripetitivo dello stesso gruppo di operazioni;

Unita' di luogo cioè: la sequenza operativa si svolge in un unico posto di lavoro.

La descrizione del task utente e' preparata dall'analista su di un modulo standard visibile in figura 1.2.

Esso contiene, sulla prima colonna, la descrizione di tutte le "azioni" eseguite dall'utente in una unita' di lavoro (INPUT) e, sulla colonna adiacente, le "azioni" corrispondenti intraprese dal sistema elaborativo (di cui una e' l'OUTPUT).

Un esempio di azione del sistema puo' essere l'accesso ad un archivio per leggere un segmento di Data Base e successivamente renderlo visibile all'operatore.

In pratica nel task utente e' contenuta non solo la descrizione del dialogo "uomo-macchina" dall'inizio del task (in figura l'operazione di "sign on") fino al suo completamento, ma anche sono raccolte tutte le azioni che intraprende il sistema a fronte di una richiesta dell'utente.

Fra le varie informazioni contenute nel diagramma di descrizione del task utente, quelle rilevanti per un programmatore sono solo le "azioni del sistema", descritte nella colonna di destra.

In chiusura di questo argomento crediamo opportuno precisare, per chiarezza, che il concetto di "task utente", e' del tutto generale e svincolato dall'ambiente CICS, tant' e' vero che un solo task utente potrebbe essere realizzato con piu' transazioni CICS.

1.4 IL PANNELLO

L'interscambio di "informazioni" che avviene tra l'operatore ed il sistema (o applicazione) e' realizzato di solito con l'ausilio di un terminale video.

Per comodita' di lettura tali informazioni vengono inserite in un tracciato grafico già predisposto; definiamo "pannello" l'insieme dei dati (o informazioni) e del loro formato visibile al terminale.

Avvalendoci di questa definizione possiamo quindi dire che il dialogo operatore-sistema avverrà esclusivamente attraverso un susseguirsi di pannelli.

Un'applicazione interattiva, solitamente, non si limita a svolgere solo funzioni di lettura e/o memorizzazione dei dati evidenziati su di un pannello, ma a volte necessita di eseguire dei calcoli sui dati in input e in output.

Per questo il task utente viene ulteriormente documentato, per ogni pannello che abbia necessità di "calcolo", approntando un modulo contenente le specifiche delle operazioni da eseguire sui dati (vedi un esempio in figura 1.3).

In particolare, i dati possono essere manipolati:

- dopo essere stati letti da un archivio e prima di essere visualizzati,
- dopo essere stati immessi dall'operatore e prima di essere scritti in archivio,
- per rilevare e notificare eventuali errori.

1.5 IL FLUSSO DEI PANNELLI

E' importante sottolineare il fatto che l'operatore, durante il suo lavoro a terminale, vede l'applicazione solo attraverso la sequenza dei pannelli e niente altro. Di qui l'importanza chiave che riveste una buona progettazione dei pannelli e la loro organizzazione all'interno del flusso operativo.

Abbiamo visto, al paragrafo precedente, qual'e' il primo documento consegnato dall'analista al programmatore: la Descrizione del Task Utente.

Ora possiamo esaminare il secondo documento che e' il "Diagramma di Flusso dei Pannelli" (vedi figura 1.4). Per far questo abbiamo ancora bisogno di qualche definizione.

Un flusso di pannelli puo' essere di due tipi:

- a) una sequenza semplice (vedi figura 1.5);
- b) una sequenza gerarchica (vedi figura 1.6).

1.5.1 La sequenza semplice

Come si vede nella figura 1.5, un flusso di pannelli in sequenza semplice e' tale quando si puo' scorrere la sequenza, dall'inizio alla fine, solo in avanti; cioe' si puo' passare solo dal primo pannello al secondo e cosi' via.

In pratica non sono possibili scelte del pannello successivo, sia in avanti sia all'indietro.

Un esempio valido di utilizzo di tale tipo di flusso puo' essere un'applicazione che faccia solo raccolta massiccia di dati (data entry).

1.5.2 La sequenza gerarchica

Come si puo' vedere nella figura 1.6, un flusso di pannelli in sequenza gerarchica e' una struttura ad albero rovesciato che presenta diversi rami, ai quali e' possibile arrivare sia per scelta operativa sia per scelta applicativa (o di sistema) e lungo i quali e' possibile muoversi sia in avanti che all'indietro.

In questo tipo di sequenza spesso e' utile fare uso di un particolare "pannello di scelta", detto "MENU", che indichi le varie possibilta' offerte dall'applicazione e che, quindi, aiuti l'operatore a scegliere il suo "ramo".

Inoltre, dato che in una struttura ramificata, le possibilita' di scelta si presentano anche durante lo svolgimento dell'applicazione, e' possibile fare uso di piu' pannelli di menu all'interno della stessa applicazione (vedi figura 1.6); oppure fornire possibilita' di scelta all'operatore nell'ambito di un pannello di immissione dati, senza fare uso di un pannello di menu separato (ad esempio indicare sull'ultima riga di ogni pannello le scelte possibili).

1.6 CONTROLLO DEL FLUSSO DEI PANNELLI

Per potersi muovere facilmente all'interno di una sequenza gerarchica, rappresentante un flusso di pannelli, e' bene che tutti i pannelli abbiano evidenziati i "legami operativi" che consentano di passare da uno di essi ad un altro qualsiasi.

Un esempio di questi legami operativi e' visibile nella figura 1.4, dove sono indicate le varie "chiavi funzionali" che sono messe a disposizione dell'operatore per effettuare le sue scelte.

Sara' cura del programmatore codificare l'applicazione in modo che gestisca l'uso di queste chiavi funzionali, in accordo con quanto richiesto dal diagramma di flusso dei pannelli.

le chiavi funzionali descritte nella figura 1.4 indicano alcuni tasti particolari di una tastiera IBM 3270. Per questo essi sono chiamati "tasti funzionali".

Nel prossimo capitolo vedremo in dettaglio le caratteristiche di una tastiera IBM 3270.

E' bene notare che l'indicazione dell'uso dei tasti funzionali nel diagramma di flusso dei pannelli e' indispensabile per stabilire degli standards operativi.

Per esempio, nella figura 1.4, il tasto PF9, se digitato, permette il ritorno al pannello di menu iniziale indipendentemente dal punto cui era l'applicazione. Questa funzione del tasto PF9 potrebbe essere generalizzata per tutte le applicazioni.

Le chiavi funzionali, comunque, potrebbero essere anche delle "parole chiave" che siano opportunamente gestite dal programma.

Nell'esempio di figura 1.4, per passare dal pannello di menu a quello iniziale di inquiry, deve essere utilizzato il tasto funzionale PF2. Nulla vieta che il pannello di menu preveda, invece (o anche), uno spazio sullo schermo dove sia possibile digitare una parola chiave, ad esempio "INQ", che, opportunamente intercettata dal programma, svolga la stessa funzione di PF2.

Quanto descritto nel diagramma di flusso, riguardo ai tasti funzionali, e' possibile trasferirlo in forma piu' leggibile su di una tabella riepilogativa, di cui la figura 1.7 e' un esempio.

1.6.1 Controllo del flusso in caso di errore

Perche' un'applicazione lavori correttamente, deve esistere sempre un tipo di controllo del flusso dei pannelli che dipenda dall'esito dell'elaborazione delle richieste inviate all'elaboratore.

In pratica vogliamo controllare l'esecuzione dell'applicazione, non solo quando l'operatore digita tasti funzionali e/o parole chiave prevista, ma anche quando accade una situazione di "errore".

Questa condizione di errore puo' avvenire in due casi distinti:

1. Quando l'operatore usa un tasto funzionale o parola chiave non prevista dal programma;
2. Quando l'esecuzione della richiesta (per esempio, l'accesso e la lettura di un record su un file) de' un esito negativo.

In entrambi i casi sopra citati, sara' compito del programma gestire la situazione anomala e prevedere il ritorno al pannello adeguato per un corretto proseguimento, con eventuale messaggio informativo per l'operatore.

1.7 LA STRUTTURA DEI DATI

Dall'esame del Diagramma del Task e del relativo Diagramma di Flusso dei Pannelli, risulta chiaro che non e' ancora possibile codificare l'applicazione, se prima non conosciamo la struttura dei dati che dobbiamo utilizzare.

E' facile rappresentare graficamente quali sono i dati in INPUT ed in OUTPUT alla nostra applicazione; basta un disegno come in figura 1.8, dove

- A1 rappresenta un archivio (File, Data Base, Coda Sequenziale) dal quale sono effettuate le letture richieste dall'applicazione;
- D1 rappresenta un documento (per esempio un ordine) dal quale l'operatore preleva i dati da inviare all'applicazione;
- A2 rappresenta un archivio (nella maggior parte dei casi A1) sul quale effettuare le scritture richieste dall'applicazione;
- D2 rappresenta il documento finale dell'applicazione.

La figura 1.8 equivale ad una tabella a due colonne, in cui sono elencate le risorse in Input e, rispettivamente, quelle in Output all'applicazione. Una tabella siffatta prende il nome di "TABELLA DI I/O".

La figura 1.8 (e la tabella equivalente) rappresenta una situazione "tipo" facilmente generalizzabile. Infatti, secondo le specifiche dell'applicazione, puo' essere modificato il numero delle risorse, sia in input che in output.

L'importante e' individuare, dalla descrizione del task utente, tutti gli archivi e i documenti necessari all'esecuzione dell'applicazione e, poi, conoscere tutti i tracciati dei record e/o dei segmenti degli archivi coinvolti.

Per questo scopo sono a disposizione del programmatore quattro tipi di documenti, uno per i file di tipo tradizionale (SAM, VSAM, ISAM) e tre per i Data Bases DL/I.

1.7.1 Files tradizionali

Il tracciato dei record per un file tradizionale e' riportato su di un modulo visibile in figura 1.9.

In esso e' specificato, tra l'altro:

- il tipo di file,
- la posizione dei campi,

- la lunghezza dei campi,
- il tipo dei dati (alfabetico, numerico, carattere, packed),
- la descrizione del campo.

1.7.2 Data Bases DL/I

Per codificare l'applicazione in ambiente DL/I, non e' sufficiente avere la descrizione dei segmenti dei record del data base, ma serve anche la descrizione della struttura gerarchica dei dati (VISTA FISICA) e della "VISTA LOGICA", cioè della parte di struttura fisica che l'applicazione deve "vedere".

Per questo sono a disposizione del programmatore tre tipi di moduli: uno per la vista fisica, uno per la vista logica ed uno per ogni segmento della vista logica.

Nella figura 1.10 e' riportato un modulo per la descrizione di un segmento e, come si puo' vedere, esso e' analogo a quello di figura 1.9, che descrive il record di un file tradizionale.

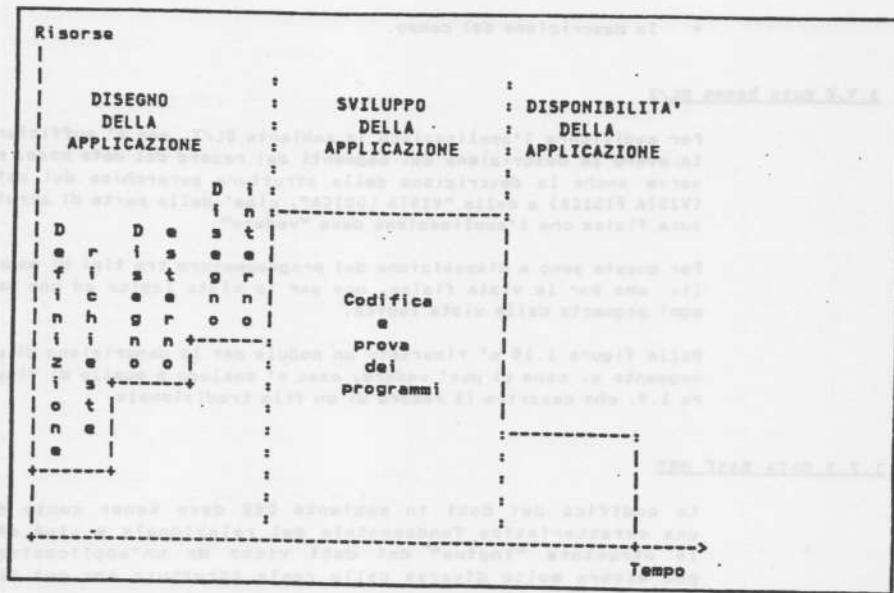
1.7.3 DATA BASE DB2

La codifica dei dati in ambiente DB2 deve tener conto di una caratteristica fondamentale del relazionale e cioè che la struttura "logica" dei dati vista da un'applicazione può essere molto diversa dalla reale struttura con cui DB2 memorizza i dati.

Per accedere ai dati DB2 all'interno di una applicazione si deve quindi fare ricorso a delle variabili particolari come vedremo nel capitolo 9.

La descrizione dei dati memorizzati da DB2 avviene per tabelle divise in righe e colonne secondo uno schema simile a quello proposto in figura 1.13.

1.9 FIGURE



**Figura 1.1 - Realizzazioni di un'applicazione:
tempi e risorse necessari.**

D E S C R I Z I O N E D E L T A S K U T E N T E

| | | | |
|------------------|------------------------|---------|----------|
| PROGETTO: | Personale | DATA | 06/12/83 |
| ATTIVITA': | Assunzione | PAGINA | 1 |
| NAME DEL TASK: | Dati nuovo impiegato | ----- | |
| EVENTO INIZIALE: | Richiesta formalizzata | ID.: | UT12 |
| DOCUMENTI INPUT: | Modulo di assunzione | AUTORE: | M.Devino |
| LOCAZIONE: | Amministrazione | FREQ.: | 15/mese |

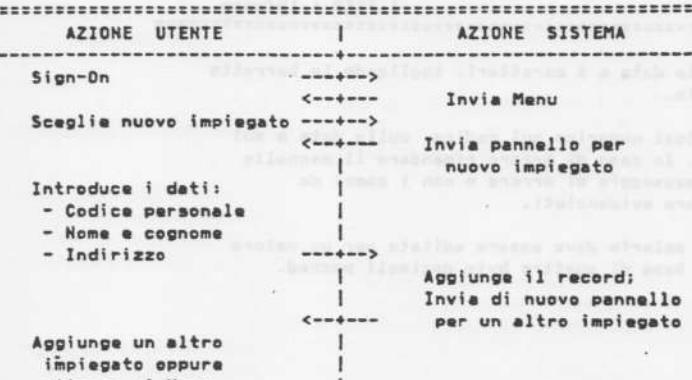


Figura 1.2 - Modulo di descrizione del Task Utente

MODALITA' DI CALCOLO

| | | |
|-------------------------------------|-----------------|------------------|
| PROGETTO: | Personale | DATA 06/12/83 |
| ATTIVITA': | Assunzione | PAGINA 1 |
| NOME DEL TASK: Dati nuovo impiegato | | +----- |
| NOME DEL PANNELLO: | ADDPAG | ID.: UT12 |
| LOCAZIONE: | Amministrazione | AUTORE: M.Devino |
| | | FREQ.: 15/mese |

1. Portare la data a 6 caratteri, togliendo le barrette divisorie.
2. Fare un test numerico sul codice, sulle date e sul salario. In caso di errore rimandare il pannello con un messaggio di errore e con i campi da correggere evidenziati.
3. Il campo salario deve essere editato per un valore sul Data Base di quattro byte decimali packed.

Figura 1.3 - Modulo per le specifiche di calcolo.

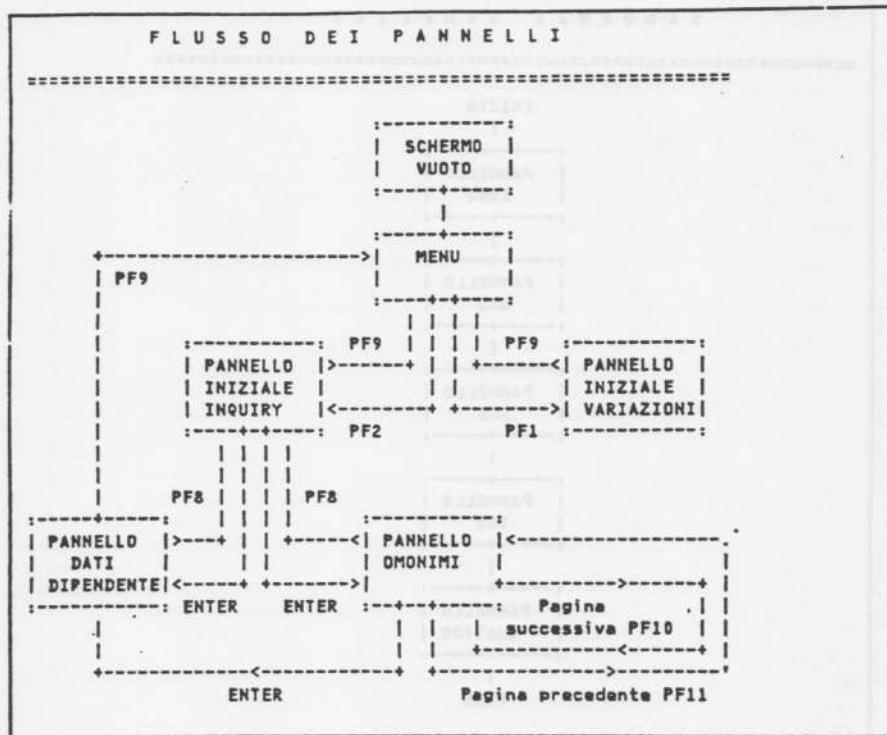


Figura 1.4 - Esempio di flusso di pannelli.

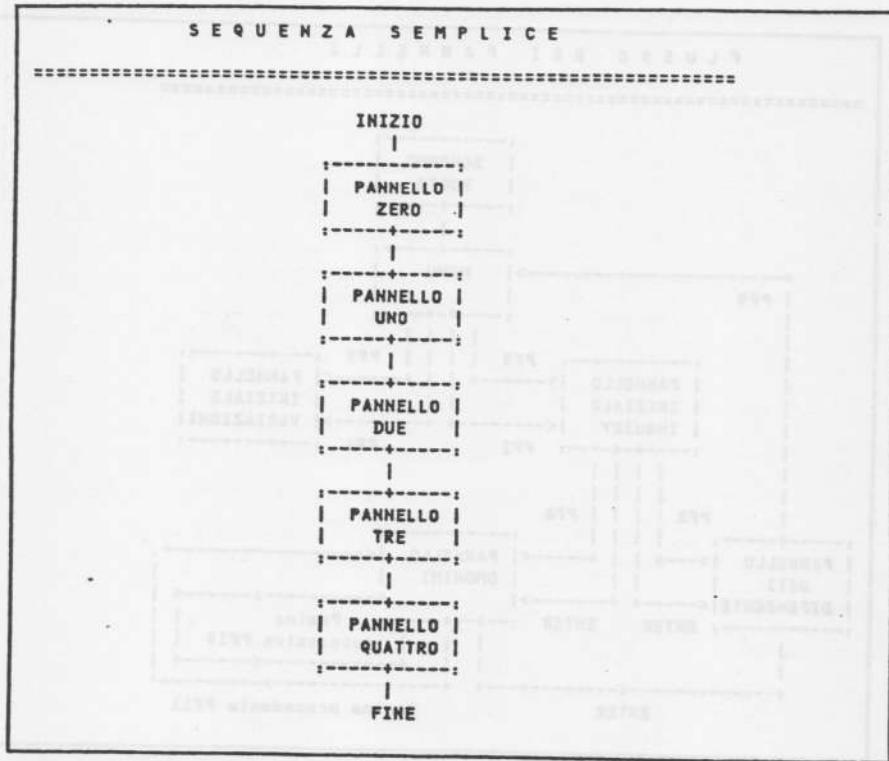


Figura 1.5 - Sequenza semplice di pannelli.

SEQUENZA GERARCHICA

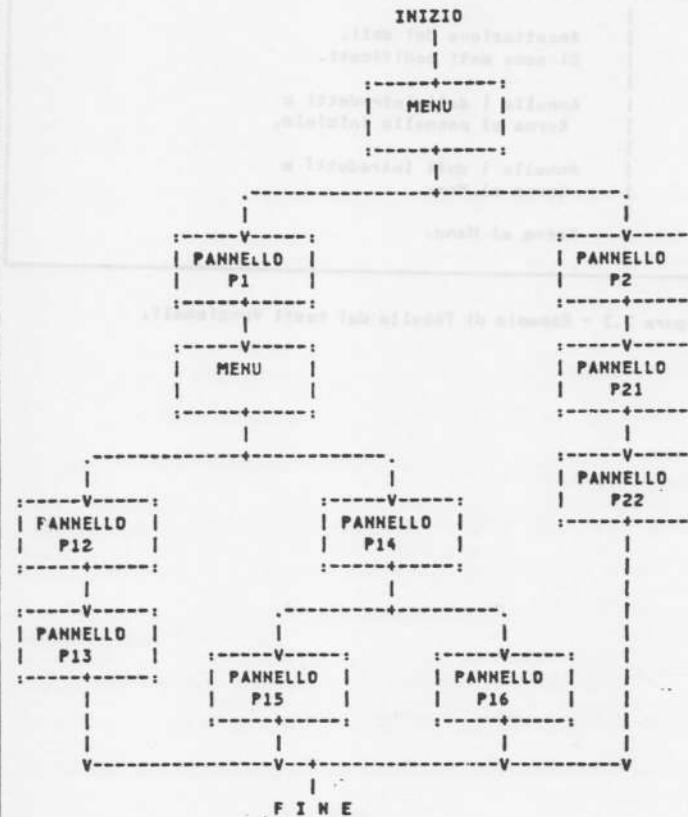
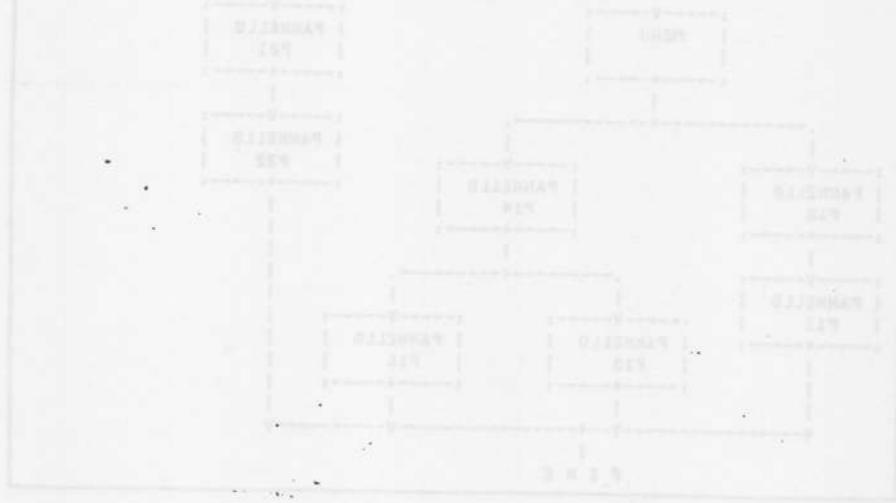


Figura 1.6 - Esempio di sequenza gerarchica di pannelli.

| TASTO | FUNZIONE |
|-------|--|
| ENTER | Accettazione dei dati. Ci sono dati modificati. |
| PF10 | Annulla i dati introdotti e torna al pannello iniziale. |
| PF11 | Annulla i dati introdotti e torna al Menu. |
| PF09 | Torna al Menu. |

Figura 1.7 - Esempio di Tabella dei tasti funzionali.



Il menù di accettazione definito nel campo 7 di CICS è:

TABELLA DI INPUT/OUTPUT

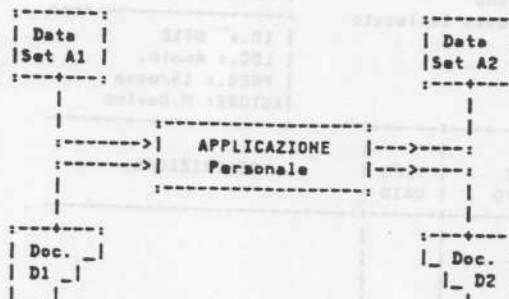


Figura 1.8 - Esempio di Tabella di I/O.

D E S C R I Z I O N E R E C O R D

| | | |
|-------------------|----------------------|------------------|
| PROGETTO: | Personale | DATA 06/12/83 |
| ATTIVITA': | Assunzione | PAGINA 1 |
| NAME DEL TASK: | Dati nuovo impiegato | |
| TIPO DEL FILE: | | ID.: UT12 |
| NAME DEL FILE: | | LOC.: Ammin. |
| LUNGHEZZA RECORD: | | FREQ.: 15/mese |
| NUMERO DI RECORD: | | AUTORE: M.Devino |

| POSIZIONE | | TIPO | DESCRIZIONE |
|-----------|-------|-------|-------------|
| DA | LUNG. | CAMPO | DATO |

| | |
|------------|--------------------|
| TIPO DATO: | A = Alfabetico |
| | C = Carattere |
| | N = Numerico |
| | P = Decimal Packed |

Figura 1.9 - Modulo di descrizione del Record.

| DESCRIZIONE SEGMENTO | | | | |
|-------------------------------------|------------|------------------|------|-------------|
| PROGETTO: Personale | | DATA 06/12/83 | | |
| ATTIVITA': Assunzione | | PAGINA 1 | | |
| NAME DEL TASK: Dati nuovo impiegato | | AUTORE: M.Devino | | |
| NAME DEL DATA BASE: | | ID.: UT12 | | |
| NAME DI DBD: | | LOC.: Ammin. | | |
| NAME DI SEGMENTO: | | FREQ.: 15/mese | | |
| POSIZIONE | NOME CAMPO | TIPO DATO | SEQ. | DESCRIZIONE |
| INIZIO LUNG. | | | | |
| : | : | : | : | : |
| : | : | : | : | : |
| : | : | : | : | : |
| : | : | : | : | : |
| : | : | : | : | : |
| | | | | |

TIPO DATO: A = Alfabetico
 C = Carattere
 N = Numerico
 P = Decimal Packed

Figura 1.10 - Modulo di descrizione dei Segmenti.

STRUTTURA GERARCHICA - VISTA FISICA

| | | |
|-------------------------------------|------------|------------------|
| PROGETTO: | Personale | DATA 06/12/83 |
| ATTIVITA': | Assunzione | PAGINA 1 |
| NOME DEL TASK: Dati nuovo impiegato | | + |
| NOME DEL DATA BASE: | | ID.: UT12 |
| NOME DBD INDICE: | | LOC.: Ammin. |
| NOME DBD FISICA: | | FREQ.: 15/mese |
| DIMENSIONE MEDIA RECORD: | | AUTORE: M.Devino |
| NUMERO STIMATO DI RECORD: | | |

DIAGRAMMA

Figura 1.11 - Modulo di descrizione della "Vista Fisica" di Data Base.

S T R U T T U R A G E R A R C H I C A - V I S T A L O G I C A

| | | |
|-------------------------------|----------------------|------------------|
| PROGETTO: | Personale | DATA: 06/12/83 |
| ATTIVITA': | Assunzione | PAGINA 1 |
| NAME DEL TASK: | Dati nuovo impiegato | +----- |
| NAME DEL DATA BASE: | | ID.: UT12 |
| NAME DEL PSB: | | LOC.: Ammin. |
| NAME DBD FISICA: | | FREQ.: 15/mese |
| LUNGHEZZA CHIAVE CONCATENATA: | | AUTORE: M.Devino |

D I A G R A M M A

| | | |
|------------|------|-----------|
| SEARCH | STAO | REGISTER |
| EDIT | SMIT | TRANSLATE |
| CHATTRSHIT | | LISTDDN |
| SEND | | TAUDIN |

Figura 1.12 - Modulo di descrizione della "Vista Logica" di Data Base.

DESCRIZIONE RECORD

| | | |
|-----------------|----------------------|------------------|
| PROGETTO: | Personale | DATA 06/12/83 |
| ATTIVITÀ: | Assunzione | PAGINA 1 |
| NOME DEL TASK: | Dati nuovo impiegato | |
| NOME DATA BASE: | | ID.: UT12 |
| NOME TABELLA: | | LOC.: Ammin. |
| PROPRIETARIO: | | AUTORE: M.Devino |

| NOME CAMPO | TIPO DATO | DESCRIZIONE |
|------------|-----------|-------------|
| | | |
| | | |
| | | |
| | | |

| | | | |
|------------|----------|-----------|---------|
| TIPO DATO: | INTEGER | DATE | GRAPHIC |
| | SMALLINT | TIME | |
| | DECIMAL | TIMESTAMP | |
| | FLOAT | CHAR | |

Figura 1.13 - Modulo di descrizione per una tabella DB2

2.1 INTRODUZIONE

Quando si sviluppa un programma interattivo, il programmatore si trova a dover utilizzare il terminale come strumento di input-output.

D'altra parte e' anche vero che a seconda del terminale utilizzato variano le possibili e le caratteristiche, basti pensare alle differenze tra uno schermo a colori ed uno monocromatico.

Nel prossimo capitolo vedremo che un componente del CICS/VIS, il BMS, permetterà al programmatore di predisporre con facilità il formato dei dati che appaiono sullo schermo, in modo indipendente dal tipo di terminale utilizzato.

Prima di affrontare il BMS, e' perciò opportuno prendere familiarità con i terminali che dobbiamo in seguito utilizzare.

Per questo il presente capitolo descriverà le caratteristiche della serie di terminali 3270.

2.2 IL TERMINALE IBM 3270

Il termine "terminale IBM 3270" nella sua accezione completa indica un sistema di visualizzazione dell' informazione costituito da tre componenti fondamentali.

Questo sono:

- L'Unita' di Controllo.
 - Il Terminale Video, che a sua volta e' composto da uno schermo e da una tastiera.
 - La Stampatrice.

Ognuna di queste componenti e' disponibile in vari modelli, al variare del modello cambiano alcune caratteristiche.

In particolare il Terminale Video puo' variare per il numero di caratteri presentabili sullo schermo, per la disponibilita' di gestire attributi estesi (per es. il colore) e per la configurazione della tastiera.

Di queste tre componenti tratteremo solo il Terminale Video, perche' e' quello che ci interessa per realizzare la mappa BMS.

2.2.1 La Tastiera

Nella figura 2.1 si vede come si presenta una tastiera 3278, senz'altro rappresentativa della serie 3270, essendo una delle piu' utilizzate.

I tasti sono raggruppabili funzionalmente in tre categorie:

- Tasti che servono per digitare i dati da parte dell'operatore.
- Tasti che servono per controllare l'immissione a terminale.
- Tasti funzionali che servono per inviare dati all'elaboratore.

Tasti di digitazione dati:

Questi (vedi la figura 2.2) occupano la parte centrale della tastiera, e comprendono i caratteri numerici, alfabetici, di interpunkzione e simboli speciali.

Quando si preme uno di questi tasti, il carattere corrispondente compare sullo schermo nella posizione indicata da un carattere mobile di sottolineatura, detto "cursore". Il tasto 1, detto "barra di spaziatura", immette sullo schermo un carattere di spaziatura che sostituisce gli eventuali caratteri già presenti.

Tasti di controllo immissione:

Questi non sono raggruppati fisicamente sulla tastiera, ma sono evidenziati nella figura 2.3. Seguendo la numerazione della figura, diamo ora una breve spiegazione del loro utilizzo:

- Tasti 1: Servono a posizionare il cursore senza modificare quanto già visualizzato sullo schermo.
- Tasto 2: È il tasto di "scambio": e serve per permettere la digitazione del carattere indicato nella parte superiore dei tasti con doppio carattere.

In particolare per i tasti alfabetici, attiva le maiuscole.

- Tasto 3: È il tasto di "blocco" e serve a bloccare la tastiera nella condizione di maiuscolo. Per rilasciare il blocco e riportare la tastiera nella posizione minuscolo, si deve premere uno dei due tasti di scambio.

I tasti 2 e 3 sono analoghi a quelli di una normale macchina da scrivere, e funzionano nello stesso modo.

- Tasto 4: È il tasto di scambio "alternato". Serve per permettere di attivare i tasti di funzione alternata. Questi

sono quelli che hanno un simbolo o un nome nella parte frontale.

- Tasto 5: E' il tasto di "inserimento". Serve per inserire uno o piu' caratteri in mezzo ad altri gia' presenti sullo schermo.
- Tasto 6: E' il tasto "cancellazione" di un carattere. Premendo questo tasto il carattere sottoscritto dal cursore viene cancellato.
- Tasto 7: E' il tasto di "reset". E' usato per ripristinare la tastiera da una condizione di "non immissione". In particolare serve per chiudere una operazione di inserimento (tasto 5).
- Tasto 8: Tasto di "cancellazione Input". Schiacciando questo tasto insieme al tasto 4, verrà cancellato tutto cio' che è stato immesso sullo schermo dall'operatore.
- Tasto 9: Tasto di "Cancellazione fino a fine campo". Questo tasto cancella dal video tutti i caratteri presenti alla destra di quello sottolineato dal cursore e fino ad un punto che dipende dalla suddivisione dello schermo (vedi piu' avanti il concetto di "campo").

Tasti per inviare dati all'elaboratore

Da un punto di vista hardware, ogni carattere dello schermo e' memorizzato in una cella di memoria contenuta fisicamente nel terminale. L'insieme di queste celle di memoria si chiama "buffer" o memoria di transito del terminale, la cui dimensione e' uguale al numero massimo di posizioni sullo schermo.

Percio' quando immettiamo dati da tastiera, quello che fisicamente succede, e' che il buffer del terminale si aggiorna per rispecchiare la situazione attuale dello schermo.

Il terminale quindi, per tutto il tempo in cui completiamo il testo sullo schermo, lavora senza impegnare l'elaboratore a cui e' collegato.

Se vogliamo comunicare con l'elaboratore e' necessario fare uso di uno tra i tasti appositi, detti appunto tasti di "invio".

Quando premiamo uno di questi tasti, dal terminale parla un messaggio per l'elaboratore che contiene il codice del tasto di invio premuto, seguito eventualmente dai dati che vogliamo far conoscere.

Il codice di identificazione del tasto di invio e' lungo un byte e si chiama AID (Attention IDentifier).

Nel caso in cui il terminale invia all'elaboratore anche i dati, non e' necessario che siano spediti tutti i caratteri contenuti

nel buffer del terminale. Infatti, e' sufficiente che ogni volta siano inviati solo i dati "modificati" rispetto all'ultima trasmissione.

Cio' e' del tutto trasparente al programmatore, ma dal punto di vista del sistema, significa un notevole risparmio di risorse.

Esaminiamo ora in dettaglio i tasti di invio (vedi figura 2.4):

- Tasto 10: Tasti PA1 e PA2. Sono tasti di funzione alternata e devono essere usati insieme al tasto 4. Questi tasti inviano all'elaboratore soltanto il proprio codice di identificazione.

Sono perciò particolarmente utili per restituire il controllo al programma o mettersi in comunicazione con esso.

- Tasto 11: Tasto "ENTER" o "INVIO". Questo tasto e' utilizzato per segnalare che il tasto sullo schermo e' stato completato e che si vuole trasmetterlo.

Il tasto "INVIO", infatti, spedisce all'elaboratore il proprio codice, insieme ai dati strettamente necessari per far gli conoscere l'attuale contenuto dello schermo (cioè i dati modificati rispetto all'ultima trasmissione).

- Tasto 12: Tasto "CLEAR" o "CANC IMM". E' anche questo un tasto di funzione alternata, e va usato insieme al tasto 4.

Questo tasto cancella ogni carattere sullo schermo, posiziona il cursore in alto a sinistra e, contemporaneamente, invia solo il proprio codice di identificazione all'elaboratore (AID).

- Tasti 13: Tasti "funzionali". Questi 12 tasti fanno la stessa funzione del tasto di "enter", cioè inviano ciascuno il proprio codice di identificazione, insieme ai dati "modificati".

Il programmatore puo' attribuire un significato a ciascuno di essi e, perciò, svolgere funzioni diverse a seconda delle esigenze applicative.

Per questo motivo i tasti funzionali si dicono anche "tasti di funzione programmabili".

I tasti funzionali sono opzionali, e la tua tastiera ne potrebbe prevedere fino a 24.

2.2.2 Lo schermo

Lo schermo di un terminale e' un normale tubo a raggi catodici come quello di un comune televisore. La principale differenza, per i terminali non grafici, consiste nel fatto che il pennellino elettronico puo' formare sullo schermo solo un certo numero di

simboli, ed esattamente quelli dell'insieme dei caratteri visualizzabili della tastiera.

Come si e' visto, oltre ad apparire sullo schermo, ogni carattere e' anche memorizzato nel buffer del terminale.

Quando si introduce un carattere da tastiera questo e' visualizzato nella posizione corrente sul video, evidenziata da un cursore mobile.

In questo modo si puo' lavorare sullo schermo come se fosse una sequenza indistinta di caratteri.

Se pero' esaminiamo i testi che normalmente devono finire su un terminale, ci rendiamo conto che e' piu' comodo considerare delle porzioni di schermo come logicamente legate; per esempio tutti i caratteri che formano la "testata" di un documento, o la zona di schermo in cui l'operatore deve introdurre la data del giorno.

Dal punto di vista del programmatore, perciò, sorge la necessita' di suddividere lo schermo in gruppi di caratteri contigui e delimitati, che chiameremo "campi".

Un campo e' quindi definito come una sequenza di caratteri a partire da un certo punto dello schermo e che si estende per una lunghezza ben definita.

Ad ogni campo e' sempre associato un byte "attributo" che ne descrive le caratteristiche tra quelle permesse dal tipo di terminale, e che si applica ad ogni carattere di cui e' costituito il campo.

Per esempio l'"attributo" di un campo comprende l'intensita' luminosa, il colore, il lampeggio, ed altre ancora, che vedremo in dettaglio in seguito.

Per i terminali 3270 la memorizzazione dell' attributo di campo occupa il primo byte del campo, e cio' significa che in quella posizione non sara' visualizzabile alcun carattere.

Dovendo definire un campo di sette caratteri, ad esempio, terremo a disposizione 9 caratteri sullo schermo: il primo carattere conterra' l'attributo di campo appropriato, i successivi 7 bytes sono a disposizione per contenere i dati di quel campo, e l'ultimo byte contiene l'attributo del campo successivo.

In ogni caso e' regola che un campo si estenda dal suo byte attributo fino al byte attributo successivo escluso.

Un altro dispositivo accessorio supportato dal terminale 3270 e' la "penna di selezione", detta anche "light-pen" o "penna luminosa".

La penna luminosa serve per selezionare un campo sullo schermo semplicemente puntando la penna su un carattere di tale campo e premendone la punta contro il video.

Infatti, la punta della penna contiene un dispositivo in grado di rilevare la luce di un carattere sullo schermo, e di riconoscerne la posizione nel buffer.

Perche' un campo sia selezionabile da penna luminosa deve essere innanzitutto definito tale assegnandogli un particolare attributo. Inoltre il primo carattere di un campo selezionabile da penna, detto "carattere designatore", deve avere un valore convenzionale:

- il punto interrogativo: ?
- la "E commerciale": &
- un carattere "blank"

Il carattere designatore "?" consente una selezione ritardata.

I caratteri designatori "&" e "blank" consentono una selezione immediata.

La scelta di un campo selezionabile da penna puo' essere effettuata anche ponendo il cursore sotto il "carattere designatore" e premendo un tasto di invio.

La scelta di un campo a selezione ritardata, provoca la modifica del carattere "?" nel carattere ">".

Solo quando l'operatore preme un tasto di invio (ENTER o PFxx), verranno inviati all'elaboratore gli indirizzi dei campi da lui scelti.

Con la selezione immediata, invece, l'indirizzo di campo viene inviato all'elaboratore ad ogni selezione. Inoltre, nel caso di campo con carattere designatore "&", vengono inviati anche i dati di ogni campo modificato.

Il carattere attributo

Come abbiamo visto parlando dello schermo, la prima posizione di ogni campo e' sempre occupata dal relativo carattere attributo, che occupa un byte del buffer, senza perci' essere evidenziata sullo schermo.

Vediamo il significato di ciascuno degli otto bits del carattere attributo in un terminale 3278.

- Bit 0 : Non rilevante.
- Bit 1 : Sempre a 1.

- Bit 2 : "Protezione". Se questo bit e' "1" non e' possibile modificarne il contenuto da tastiera. Se l'operatore si posiziona col cursore e preme un tasto la tastiera si blocca, e non permette la modifica del campo protetto. Se il bit e' "0" il campo puo' essere normalmente modificato dall'operatore utilizzando la tastiera.

La protezione di un campo e' utile per campi di solo output, come per esempio una testata, dove l'operatore non deve poter cambiare il testo che si trova sullo schermo.

- Bit 3 : "Alfanumerico o numerico". Questo bit ha significato solo se la tastiera prevede un dispositivo aggiuntivo che distingue tra tasti numerici e non numerici. In questo caso se il bit e' "1", il campo e' definito "numerico": l'operatore non riuscira' ad introdurre nel campo dei caratteri che non siano componenti di un numero decimale, perch'e' altrimenti la tastiera si blocca e l'immissione risulta inibita.

Se questo bit e' "0" il campo e' definito "alfanumerico", e il terminale accetta in questa area un qualunque carattere.

Questo attributo e' utile per condizionare l'operatore ad introdurre in un certo campo solo dati numerici.

- Bits 4 e 5 : Questi bits insieme assumono 3 configurazioni diverse con il seguente significato:
 - "00" : Il campo verrà visualizzato normalmente.
 - "10" : Il campo verrà visualizzato ad alta intensità luminosa.
 - "11" : il campo verrà visualizzato ad intensità nulla, il che significa che non sarà visibile sullo schermo.
- Bit 6 : Sempre a zero.
- Bit 7 : Questo ultimo bit si chiama "Modified Data Tag" ovvero "Indicator di dato modificato" e si abbrevia in MDT. Questo bit e' automaticamente impostato dal terminale nel momento in cui il contenuto del campo relativo a questo attributo viene modificato dall'operatore attraverso la tastiera. E' proprio questo bit che permette di discriminare quali campi devono essere trasmessi all'elaboratore per informarlo in modo efficiente dell'attuale situazione dello schermo, come e' stato già fatto notare in un precedente paragrafo.

Il programmatore, vedremo in seguito, puo' giocare col bit MDT per forzare il terminale a trasmettere alcuni campi anche se non sono stati modificati dall'ultima trasmissione.

In questa descrizione del byte attributo, abbiamo preso come esempio un terminale monocromatico, che possiede un solo buffer.

I terminali a colori, come il 3279, hanno un secondo buffer che consente di codificare anche altri attributi, ad esempio il colore del campo.

Cio' non toglie che rimangano fissi i concetti di "campo" ed "attributo. Inoltre, ed anche piu' importante, e' il fatto che il programmatore lavora utilizzando come interfaccia software verso il terminale il BMS, che permette di definire e maneggiare attributi usando nomi simbolici, senza entrare nei dettaglio delle configurazioni dei bit.

2.3 SOMMARIO

In questo capitolo abbiamo visto le caratteristiche fondamentali del terminale 3278 IBM, che rappresenta significativamente la serie dei terminali 3270.

In particolare abbiamo esaminato la testiera, con tutti i suoi tasti per

- immettere dati nel buffer del terminale,
 - modificarli,
 - inviarli all'elaboratore,

• lo schermo, che puo' essere suddiviso in zone distinte (campi) precedute da un byte attributo

2.4 FIGURE

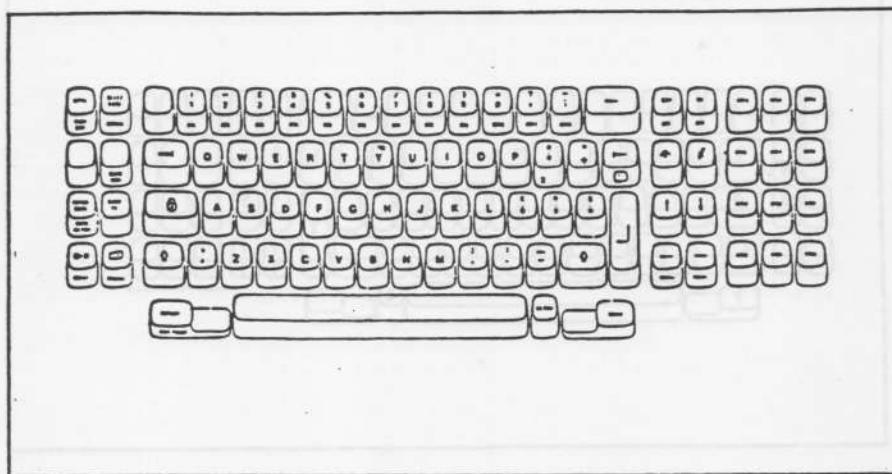


Figura 2.1 - La tastiera del terminale 3278.

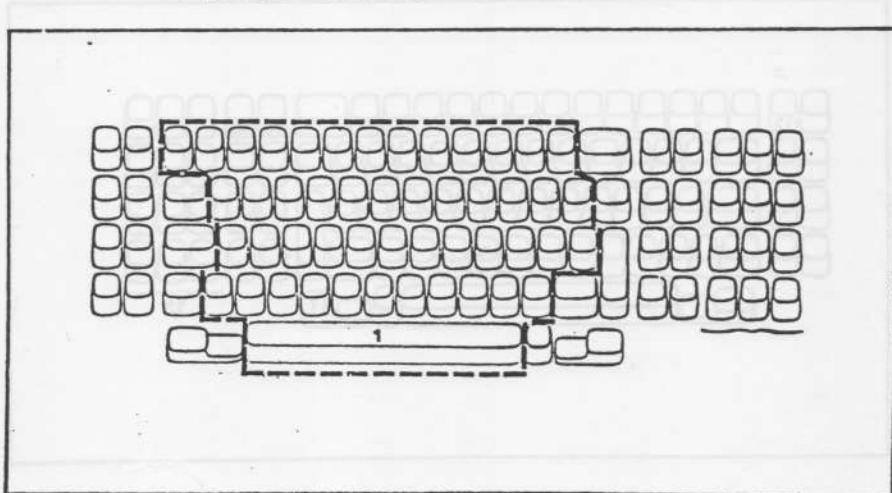


Figura 2.2 - I tasti per digitare i dati.

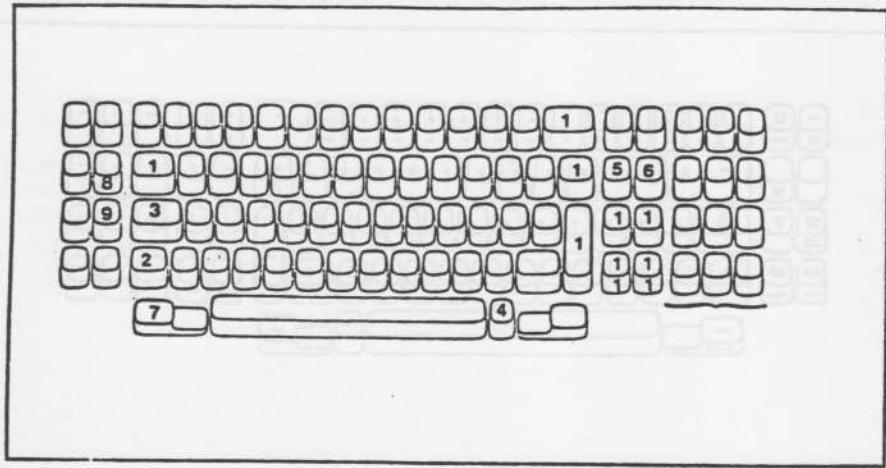


Figura 2.3 - I tasti di controllo dell'immissione.

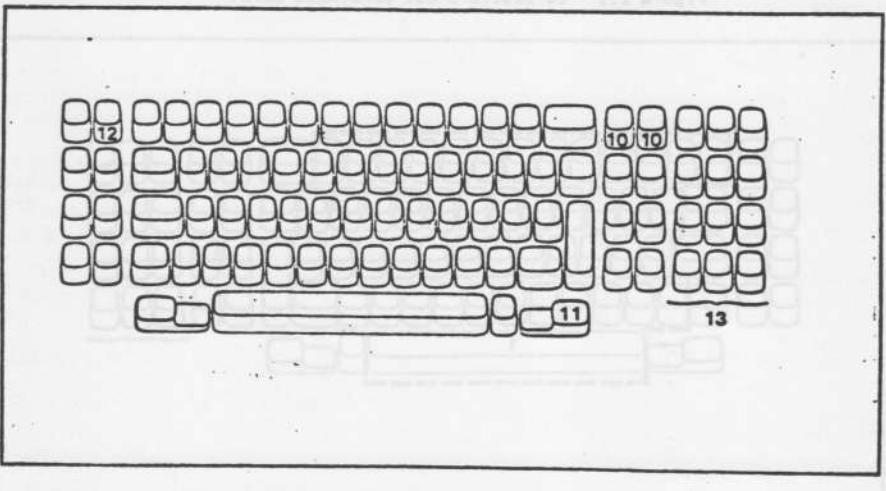


Figura 2.4 - I tasti per l'invio dei dati.

PARTE PRIMA

3.1 INTRODUZIONE

In questo capitolo studieremo lo strumento che il CICS/V5 mette a disposizione del programmatore per facilitarne il lavoro riguardo ai problemi di comunicazione con i terminali.

Questo strumento e' il Basic Mapping Support (BMS).

3.2 I VANTAGGI DEL BMS

Ad ogni trasmissione che avviene tra un terminale ed un elaboratore, o viceversa, e' associata sempre una "stringa di dati", piuttosto complessa, che e' rappresentata dall'insieme dei dati elaborativi e dei caratteri di controllo (oltre che della linea) del terminale interessato.

Poiche' un terminale si aspetta sempre di trasmettere, o ricevere, "stringhe di dati" sarebbe compito del programmatore preparare, o analizzare, queste stringhe in forma dipendente dal tipo di terminale in uso.

Questo significa che, per comunicare gli stessi dati a terminali di tipo diverso, avremmo la necessita' di scrivere tanti programmi che differiscono tra loro solo per la parte relativa ai caratteri di controllo del terminale.

Inoltre se abbiamo la necessita' di modificare il formato dello schermo, cioè "l'aspetto" che debbono assumere i dati sul terminale, dovremmo cambiare tutti i programmi interessati.

Per evitare questo lavoro dispendioso abbiamo a disposizione il BMS, che rendera' un nostro programma applicativo indipendente da:

- I tipi di terminali collegati alla rete (DEVICE INDEPENDENCE)
- I formati che debbono assumere i dati al terminale (FORMAT INDEPENDENCE)

Il BMS agisce come una "interfaccia" fra il programma applicativo e il TCP, che e' responsabile della gestione fisica dei terminali (input/output).

3.2.1 Device Independence

TP = transazione

CICS = linguaggio + transazione

Permette di scrivere il programma senza tener conto delle caratteristiche fisiche dei terminali. Il BMS, infatti, prepara le stringhe dei dati da inviare inserendo i caratteri di controllo appropriati per il terminale in uso. Per questo:

1. uno stesso programma applicativo puo' scambiare dati con terminali di caratteristiche diverse;
2. modifiche hardware e di reti non richiedono la revisione dei programmi.

3.2.2 FORMAT INDEPENDENCE

Permette di scrivere il programma senza tener conto della dislocazione fisica che occuperanno i dati su uno schermo o su un modulo prestampato.

Per utilizzare questa funzione il programma fa riferimento a "nomi simbolici" che referenziano i campi da riempire. Il BMS, correla poi questi nomi simbolici con la posizione fisica dei campi sullo schermo. Come vedremo subito, per far questo il BMS si serve di due tipi di tabelle che referenziano rispettivamente la parte "fisica" e la parte "logica" della mappa.

I vantaggi che ne conseguono sono evidenti:

1. i campi modificabili a programma possono essere riordinati, per motivi ergonomici, senza che si debba necessariamente toccare il programma che li riferenzia.
2. i campi costanti (es. testate e descrizioni) possono essere riscritti e spostati senza modificare alcun programma.
3. lo stesso insieme dei campi variabili (logici) possono essere utilizzati su supporti di caratteristiche diverse (video a 24 o a 43 righe).

3.2.3 CARATTERISTICHE FISICHE

La parte fisica descrive il formato "visibile" di tutti i campi che devono essere scritti o ricevuti da un certo tipo di terminale. E' costituita da un insieme di macro istruzioni che includono informazioni su:

- Lunghezza e posizione dei campi
- Attributi di campi (es. luminoso, protetto)
- Costanti (es. testate, descrizioni)

Ogni mappa fisica deve essere assemblata e catalogata nella libreria dei programmi CICS/VS.

Quando un programma (o gruppo di programmi correlati) fa uso di più mappe, di solito queste vengono raggruppate e definite come costituenti un MAPSET.

Quindi un MAPSET è un insieme di mappe (costituito al limite da una sola mappa) definite assieme per uno stesso tipo di terminale.

Un mapset è trattato dal CICS come un qualsiasi programma, (infatti possiede un'entrata in PPT) ed è caricato in memoria ogni volta che venga richiamato da un'applicazione.

Evidentemente per ogni tipo di terminale che faccia uso dello stesso formato di dati è necessario costruire una mappa fisica specifica ed il relativo mapset.

Quindi un programma che utilizzi terminali fisici diversi deve avere a disposizione anche delle Mappe Fisiche diverse.

Quindi un programma che utilizzi terminali fisici diversi, deve avere a disposizione anche delle Mappe Fisiche diverse.

Per questo il BMS personalizza il nome "generico" di un MAPSET con un suffisso di un carattere (specifico per il tipo di terminale), permettendo la scelta del mapset fisico opportuno.

Facciamo un esempio dell'uso a programma del nome "generico" di un mapset.

Nella figura 3.1, sia il task T1 che il task T2 usano lo stesso programma, ma sono associati, attraverso la propria TCT:E, a due terminali diversi: T1 ad un 3278, Modello 2, e T2 ad un 3767.

Quando il programma riferisce la mappa MAPI del mapset MAPSET1, il BMS si comporterà in modo diverso secondo che sia in esecuzione il task T1 o il task T2.

In particolare se il task è T1 e la mappa non è già in memoria, il BMS chiede che venga caricato in DSA un mapset noto al CICS/VS come MAPSET1P.

Se è in esecuzione il task T2 viene richiesto il caricamento del mapset MAPSET1P.

Vedi la figura 3.1.

3.4 LA DEFINIZIONE LOGICA (O SIMBOLICA)

Tutti i campi relativi ai dati variabili costituiscono la parte logica della mappa.

Per accedere a questi (campi) per nome, la loro definizione deve essere copiata nell'area di lavoro del programma.

I dati descritti possono essere rappresentati in tre formati:

- Formato FIELD
- Formato BLOCK
- Formato TEXT

Nel seguito faremo riferimento al formato "field" come quello piu' comunemente utilizzato e piu' facile da usare in quanto dotato della possibilita' di suddivisione dei dati in campi.

3.4.1 Il Formato FIELD

Quando si usa il formato "field" i dati vengono gestiti suddivisi in campi separati.

Ad ogni campo e' associato un nome simbolico che e' utilizzato per passare dati al BMS o per riceverli da esso.

Ogni campo e' preceduto da un prefisso di:

- tre byte, per schermi monocromatici
- sette byte, per schermi a colori

di cui i primi due contengono la lunghezza dei dati contenuti nel campo (usato in input dal BMS) ed gli altri contengono gli attributi del campo (usati in output dal programma nel caso intenda temporaneamente modificare gli attributi di campo descritti in una mappa relativa ad un 3270).

A tale scopo il BMS fornisce una lista di istruzioni che definiscono al programma la struttura dei dati da scambiare con i terminali.

Esaminiamo il caso piu' semplice di un prefisso di tre byte, cioè il caso di uno schermo monocromatico.

L'esempio di seguito riportato mostra che il campo della mappa identificato dal nome FLD1 viene suddiviso, a cura del BMS, in piu' sottocampi e a ciascuno viene associato un nome simbolico:

FLD1L individua il campo contenente la lunghezza di FLD1

FLD1A individua il byte attributo (usato dal programma per operazioni di output)

FLD1F individua il "flag-byte" (usato per operazioni di input). FLD1F e' una ridefinizione di FLD1A.

FLD1I individua l'area destinata a contenere i dati immessi per il campo FLD1 (input)

FLD10 individua l'area di memoria da cui inviare i dati da evitare di inviare a video per il campo FLD1 (output). FLD10 è una ridefinizione di FLD1, se la mappa è di input/output.

| LL | A | D A T I |
|----|---|--------------------|
| | | FLD1 oppure FLD10 |
| | | FLD1F oppure FLD1A |
| | | FLD1L |

Sia il dato di input che il corrispondente dato di output sono a disposizione del programma applicativo utilizzando il nome simbolico con suffisso 0 per l'input oppure quello con suffisso 0 per l'output.

Questo gruppo di nomi simbolici costituisce la definizione logica della mappa. Una o più mappe costituiscono un mapset. Il mapset fisico è a tutti gli effetti un programma e viene memorizzato nella libreria degli eseguibili mentre il mapset logico è memorizzato nella libreria dei moduli sorgente.

Per definire un'area di lavoro (nel programma) adatta a questi dati basta includere il mapset nel programma al momento della compilazione.

Perciò in un programma COBOL si tratta di definire uno statement COPY per il mapset in WORKING-STORAGE SECTION.

Il formato "field" è il più frequentemente usato, sia per terminali video che stampanti.

3.4.2 Il Formato BLOCK

Quando si usa il formato "block" i dati vengono gestiti suddivisi in "segmenti di linea".

E' possibile attribuire ai campi, all'interno di ogni segmento, un nome simbolico per facilitare la programmazione.

Ogni campo è preceduto da un prefisso di un byte contenente l'eventuale attributo (da usare in output con le stesse modalità viste per il formato "field").

Sequenze di caratteri blank separano i campi tra di loro, e servono a determinarne la posizione all'interno del segmento.

Il formato "block" può essere usato per tutti i tipi di terminali, ma è specialmente indicato per le operazioni di input su terminali scriventi.

3.4.3 Il Formato TEXT

Quando si usa il formato "text" i dati vengono gestiti come stringhe non differenziate in campi, al cui interno possono comparire caratteri di "a capo" (New Line).

Il BMS suddivide la stringa in linee la cui lunghezza e' funzione del tipo di terminale cui la stringa deve essere inviata.

La suddivisione avviene senza mai spezzare le parole e senza mai suddividere tra due linee consecutive. I caratteri di New Line inseriti dal programmatore vengono rispettati.

Il BMS inserisce gli appropriati caratteri di controllo per l'esecuzione fisica della operazione di a capo (salto linea, ritorno carrello, ecc.) ed elimina eventuali caratteri blank in coda al messaggio.

Se il programmatore ha inserito nella stringa eventuali caratteri di tabulazione, esso è responsabile dell'inserimento anche del numero appropriato di New Line.

Per la gestione dei dati in formato "text" non vengono usate mappe, richieste invece per i formati "field" e "block".

Il formato "text" puo' essere impiegato per gestire testi non formattati, di cui e' difficile prevedere la suddivisione in campi, come accade ad esempio nelle applicazioni di Message Switching, dove il formato del messaggio e' libero e variabile di volta in volta secondo le esigenze degli operatori.

PARTE SECONDA

3.5. RICHIAMI

Questo paragrafo è un sommario delle informazioni viste in precedenza.

Ci sembra opportuno in effetti ritornare su tali argomenti per capire a fondo il significato di "mappe" e "dsect" come comunemente vengono indicate la definizione fisica e la definizione logica di una mappa.

Mappe

Definiscono e attribuiscono un nome a tutti i campi che devono essere scritti o ricevuti da un terminale.

Una volta assemblate, contengono tutte le informazioni dipendenti dal tipo di terminale e necessarie a creare le stringhe di output ed a interpretare quelle di input.

Le mappe vengono catalogate nella libreria dei programmi CICS/V.S.

Dsect

Vengono generate utilizzando le "stesse" macroistruzioni usate per le mappe.

Producono una tabella di dichiarative di campi da copiare all'interno di ogni programma che intenda usare la mappa associata.

Mappe e Dsect vengono assemblate off-line usando macroistruzioni CICS/V.S. appropriate.

I dati di output vengono forniti al BMS dal programma applicativo che li muove nei campi definiti nelle Dsect. Si noti al riguardo che:

- le mappe di output possono contenere campi inizializzati, titoli, costanti ecc.
- il programma può sopprimere la visualizzazione dei campi inizializzati e delle costanti
- il programma può temporaneamente modificare gli attributi dei campi di una mappa di output.

I dati di input vengono forniti al programma applicativo dal BMS che li muove all'interno dei campi definiti nelle Dsect. Notiamo che:

- le mappe di input identificano i campi come numerici ed alfanumerici, ed il BMS provvede all'allineamento a destra o a sinistra e agli eventuali riempimenti (zeri o blank, rispettivamente)
- il prefisso lunghezza di ogni campo viene impostato dal BMS e contiene la lunghezza dei dati effettivamente introdotti dall'operatore nel campo.
- un indicatore (vedere nel seguito) consente di individuare quei campi modificati ma non contenenti dati.

Le operazioni di approntamento formati (edit) vengono attivate da appositi comandi.

L'uso del BMS non sostituisce l'intervento del Terminal Control nel leggere o scrivere dati su di un terminale.

3.6 MACRO PER GENERARE MAPPE

In questo corso impareremo a scrivere programmi che usano i comandi del CICS/V5, anche per quanto riguarda le funzioni di input/output del BMS.

Perciò, per creare MAPSET fisici e logici, dobbiamo far uso di macro istruzioni, scritte con le regole del linguaggio Assembler, avanti il seguente formato:

```
nome-macro MACKO PARAM1,PARAM2
                           ,PARAM3
```

Dove:

1. Il nome esterno assegnato alla macro va a colonna 1
2. Il nome fisso della MACRO va a colonna 10, e successive, e deve essere separato da almeno un blank da "nome-macro"
3. Il primo parametro deve essere separato da MACRO con un blank e dal parametro successivo con una virgola
4. Gli altri parametri devono essere separati tra loro da una virgola. Un blank segnala la fine dei parametri.
5. A colonna 72 e' permesso un carattere di continuazione dei parametri, che devono proseguire alla colonna 16 della riga successiva.

Quando si progettano applicazioni per i video, e' necessario disegnare il tracciato dell'immagine che deve comparire sullo schermo; tale compito e' proprio dell'analista, anche se spesso

viene affidato al programmatore. La scrittura dei programmi e la creazione delle mappe sono però due attività ben distinte.

Le funzioni BMS, sebbene applicabili indifferentemente a terminali video o stampanti, si riesce a descriverle più intuitivamente se ci si riferisce ad un video. Useremo perciò la terminologia relativa a questo tipo di terminale, tenendo presente che essa si trasferisce senza problemi ai terminali stampanti, e discuteremo solo le operazioni che richiedono una singola mappa per schermo.

Ricordiamo che ogni mappa fa parte (o costituisce da sola) di un MAPSET.

Come abbiamo visto in figura 3.1 ogni riferimento ad una qualunque mappa di un Mapset provoca il caricamento in memoria di tutto il Mapset per tutta la durata del task o fintantoché un altro Mapset non viene referenziato dal task.

Prima di esaminare in dettaglio come si creano le mappe, vediamo brevemente le funzioni delle differenti macro (Vedi la figura 3.2).

1. DFHMSD TYPE = MAP

Questa macro definisce un MAPSET e consente di raggruppare in una unica entità tutte le mappe necessarie alla costruzione di una pagina, più in generale di un messaggio. Nella Program Processing Table vi è una entrata per ogni mapset, ed è necessario un solo accesso alla libreria dei programmi per caricare in memoria tutte le mappe che lo costituiscono.

2. DFHMDI

Questa macro definisce la singola mappa all'interno del mapset, in particolare determina la posizione di inizio, sullo schermo, della mappa. Per messaggi costruiti con una sola mappa, assumeremo che la posizione di inizio sia la linea 1, colonna 1.

3. DFHMDF

Questa macro fornisce informazioni relative ai campi all'interno della mappa. Ogni campo è definito da una macro DFHMDF.

4. DFHMSD TYPE=FINAL

Definisce la fine di un mapset.

Lo stesso gruppo di macro istruzioni viene utilizzato per creare contemporaneamente sia la definizione fisica che la definizione logica di tutte le mappe del mapset.

Alcuni parametri di macro sono unici per definire le caratteristiche fisiche ed altri sono unici per quelle logiche.

Riassumendo è necessario una macroistruzione per definire il mapset, tante macro DFHMDI quante sono le mappe del mapset, tante macro DFHMDF quanti sono i campi di ogni mappa, una macro per chiudere il mapset.

Possiamo schematizzare in questo modo, tenuto conto che per messaggi costruiti con una unica mappa il mapset contiene solo una mappa:

DFHMSD TYPE=MAP

DFHMDI

DFHMDF
DFHMDF

DFHMSD TYPE=FINAL
END

Vediamo ora in dettaglio le macro appena nominate.

3.6.1 Definizione di un MAPSET (macro DFHMSD)

La macro DFHMSD e' usata per definire un gruppo di mappe BMS. Essa consiste di una serie di operandi che definiscono le caratteristiche delle mappe nel MAPSET.

MAPSET

1. nome DFHMSD TYPE=(DSECT/MAP/FINAL)

2. MODE=(OUT/IN/INOUT).
3. LANG=(ASM/COBOL/PLI).
4. (STORAGE=AUTO/BASE=nome).
5. TIOAPFX=(NO/YES).
6. CTRL=((PRINT)(L40/L64/L80/HONEOM)
(.FREEKB)(.ALARM)(.FRSET)).
7. EXTATT=(NO/YES/MAPONLY).
8. COLOR=(DEFAULT/BLUE/RED/PINK/GREEN
/TURQUOISE/YELLOW/NEUTRAL).
9. HIGHLIGHT=(OFF/BLINK/REVERSE
/UNDERLINE).
10. PS=(BASE/ps1d).
11. VALIDN=((MUSTFILL)(.MUSTENTER)
(.TRIGGER)).
12. (TERM-type/SUFFIX=n).

1. Nome, lungo da 1 a 7 caratteri, deve essere univocamente determinato e indica il nome generico del MAPSET.

2. MODE= indica se la mappa e' di input (IN), output (OUT) o di input/output (INOUT).
Supponiamo che FLD1 sia il nome di un campo variabile descritto in un mapset fisico. Nel corrispondente mapset simbolico

IN produce i nomi simbolici

- FLD1L (lunghezza)
- FLD1F (flag)
- FLD1I (dato di input)

OUT produce i nomi simbolici

- FLD1A (attributo)
- FLD1O (dato di output)

INOUT produce una combinazione dei nomi simbolici risultante dalla somma delle due precedenti opzioni.

3. LANG=ASM / COBOL / PL/I

Indica il linguaggio del programma che utilizza questo MAPSET. Rilevante solo per la creazione del mapset simbolico.

4. STORAGE=AUTO / BASE=nome

Indicano se le mappe del mapset occuperanno la stessa area di memoria, sovrapponendosi ognuna all'ultima caricata (BASE=nome), oppure se occuperanno aree diverse di memoria, senza sovrapporsi STORAGE=AUTO.

STORICO = lo gestisce noi (archivio)

ASP=arch.stor/potafoglio.

5. TIOAPFX=YES e' obbligatorio in ambiente CICS Comandi. Il parametro agisce solo in concomitanza della specifica TYPE=DSECT ed ha la funzione di non considerare le informazioni di controllo (i primi 12 caratteri) nella espansione della mappa simbolica. Una ulteriore delucidazione e' fornita dalla figura 3.3.
TIOAPFX=YES e' assunto automaticamente se STORAGE=AUTO.
6. CTRL= specifica le modalita' con cui eseguire l'operazione di output
 - FREEKB richiede lo sblocco della tastiera in modo da consentire all'operatore di introdurre il messaggio successivo.
 - ALARM aziona il segnale di allarme acustico sul 3270.
 - FRSET (field reset) imposta a 0 tutti i Modified Data Tag (MDT) nel buffer di un 3270.
 - PRINT, L40, L64, L80, HONEOM sono usati per specificare operazioni di stampa su stampatrici del sistema 3270 (3284, 3286, ecc.)
 - PRINT da' inizio alle operazioni di stampa. Se omesso, il messaggio arriva nel buffer della stampatrice ma non viene trasferito sulla carta;
 - L40, L64, L80 specifica la lunghezza della riga di stampa;
 - HONEOM (mutuamente esclusivo con i precedenti L40, L64, L80) indica che nel messaggio il programmatore ha inserito i caratteri di "New Line" (NL) e "fine messaggio" (EOM) e che essi devono essere onorati.
7. EXTATT= specifica che il BMS supporta l'attributo esteso (campo di quattro byte indicante, nell'ordine, colore, alta intensita' luminosa, caratteri programmabili e controllo dell'input) per i terminali video a colori.
 - NO l'attributo esteso non e' supportato
 - MAPONLY indica la validita' dell'attributo esteso; il programma applicativo non puo' pero' modificare il contenuto dei campi ad esso riferiti.
 - YES indica che l'attributo esteso puo' essere specificato in una mappa e che il programma applicativo puo' modificarlo dinamicamente.
8. COLOR= specifica un colore standard da assumere per tutti i campi delle mappe in un mapset. Puo' essere cambiato attraverso una diversa opzione COLOR codificata nelle macro DFHMDI e DFHMDF. E' ignorato se EXTATT=NO.

9. HIGHLIGHT= indica l'opzione intensita' luminosa per tutti i campi del mapset

- OFF l'intensita' luminosa non viene usata
- BLINK i campi sono lampeggianti
- REVERSE i campi sono evidenziati con una vista negativa della loro immagine
- UNDERLINE i campi sono visualizzati con settolineatura

10. PS= specifica che possono essere usati i "caratteri programmabili". Ovvero e' possibile programmare i singoli punti luminosi costituenti un carattere dello schermo ed ottenere cosi' "forme" diverse dai normali caratteri alfanumerici, utili per costruire grafici e disegni.

- BASE uso del set di caratteri standard
- PSid uso di un gruppo di caratteri programmati dall'utente identificabile attraverso un singolo carattere EBCDIC oppure nella forma X 'nn'

11. VALIDIN= controlla i campi in fase di immissione. Questo operando e' valido per un terminale 8775 con le feature di variazione.

- MUSTFILL specifica che il campo di input deve essere digitato per tutta la sua lunghezza, il cursore non puo' essere spostato da un campo parzialmente immesso; il dato non puo' essere trasmesso se incompleto.
- MUSTENTER indica l'obbligo di immettere almeno un carattere nel campo e che esso non puo' essere saltato dal cursore.
- TRIGGER indica un campo inviabile al programma senza l'uso di un tasto di invio dati. Se il campo e' stato appena modificato e si tenta di rimuoverne il cursore, i dati sono automaticamente trasmessi al programma.

12. TERM=tipo / SUFFIX=n
Indica il suffisso da assegnare al mapset

- in base al tipo di terminali;
- specificando un carattere particolare in SUFFIX.

La versione MINIMA del BMS supporta solo terminali 3270. Percio' sono possibili solo due valori per TERM:

- TERM=3270-1, (di 40 colonne)

• TERM=3270-2, (di 80 colonne)

con le mappe suffissate L e M rispettivamente.

Riassumendo

1. Un mapset puo' contenere una o piu' mappe
2. Ogni mapset richiede una entrata in PPT
3. I mapset vengono catalogati nella libreria dei programmi sotto un nome che "include" il carattere di suffisso legato al tipo di terminale. Il programma applicativo, quando in un comando BMS riferenzia un mapset, utilizza solo la prima parte del nome di catalogazione, escludendo cioè il suffisso. Il suffisso viene determinato in fase di esecuzione dal BMS in base alle informazioni contenute nella TCTTE.

3.6.2 Definizione di una Mappa (macro DFHMDI)

La macro DFHMDI serve per nominare e descrivere una mappa.

In un mapset ci devono essere tante macro DFHMDI quante sono le mappe.

MAPPE

1. nome DFHMDI
2. SIZE-(line,colonne)
3. LINE-numero,
4. COLUMN-numero,
5. JUSTIFY-(LEFT/RIGTH),
6. TIOAPFX-(NO/YES),
7. CTRL- come nella macro DFHMSD,
8. EXTATT- come nella macro DFHMSD,
9. COLOR- come nella macro DFHMSD,
10. HIGHLIGHT- come nella macro DFHMSD,
11. PS- come nella macro DFHMSD,
12. VALIDN- come nella macro DFHMSD.

La macro DFHMDI puo' essere usata per modificare le specifiche per TIOAPFX, CTRL, EXTATT, COLOR, HIGHLIGHT, PS, e VALIDN della macro DFHMSD.

In questa macro i parametri hanno il significato seguente:

1. Nome (da 1 a 7 caratteri) obbligatorio della mappa all'interno del mapset
2. SIZE fornisce le dimensioni della mappa in termini di lunghezza (numero di linee orizzontali) e larghezza (numero di

caratteri, o colonne, per linea). Le linee e le colonne vanno da 1 a 240.

3. LINE specifica la linea della pagina da cui parte la mappa
4. COLUMN specifica la colonna della pagina a cui deve essere allineata la mappa.
5. JUSTIFY specifica che le coordinate di partenza della mappa, date con LINE e COLUMN, devono intendersi calcolate a partire dal margine destro o sinistro della pagina. Se nulla e' specificato, viene assunto il margine sinistro (JUSTIFY=LEFT)
6. TIOAPFX=YES deve essere specificato in ambiente CICS Comandi. Se non viene specificato, resta valido l'analogo parametro codificato nella macro DFHMSD.

3.6.3 Definizione dei Campi (macro DFHMDF)

Sappiamo che, prima di codificare le macro istruzioni per generare una mappa, e' necessario disegnare (o avere a disposizione) il pannello relativo, cioe' l'immagine che deve apparire sullo schermo.

In particolare dobbiamo conoscere le caratteristiche di tutti i campi della mappa da definire, e cioe':

- dove posizionare il cursore;
- dove devono iniziare i campi nella mappa (notiamo, non sullo schermo);
- la lunghezza massima dei dati che vi possono essere contenuti;
- se tale lunghezza massima debba essere logica o fisica, cioe' se occorre creare una delimitazione fisica di fine campo;
- se il campo e' modificabile da tastiera, cioe' se e' protetto o non protetto;
- se il campo e' numerico;
- se contiene dati costanti.

Note

Poiche' un campo, per definizione, si estende dal suo attributo fino all'attributo del campo successivo, e' bene che un campo modificabile sia fisicamente delimitato, per limitarne il numero dei caratteri digitabili.

Per questo viene di solito definito un "campo delimitatore" o "tappo" (con attributo di "autoskip") detto adiacente all'ultima posizione digitabile del campo precedente.

Per ogni campo della mappa deve esserci una macro DFHMDF, i cui parametri sono:

CAMPPI

1. nome DFHMDF
2. POS=(numero/(linea,colonna))
3. LENGTH=numero,
4. ATTRB=(ASKIP/PROT/UNPROT) (.NUM).
 (.BRT/NORM/DRK) (.IC) (.DET)
 (.FSET))
5. (INITIAL=costante/XINIT=dati-exa).
6. JUSTIFY=(LEFT/RIGTH), (BLANK/ZERO)).
7. GRPNAME=nome di gruppo.
8. OCCURS=numero.
9. PICIN=valore, PICOUT=valore.
10. COLOR= come nella macro DFHMSD.
11. HIGHLIGHT= come nella macro DFHMSD.
12. PS= come nella macro DFHMSD.
13. VALIDN= come nella macro DFHMSD.

Alcuni commenti ai parametri contenuti nelle macro:

1. Una definizione per ciascun campo. Nelle mappe che devono essere usate solo in input, e' necessario definire solo i campi che verranno referenziati dal programma. Per tutti i tipi di mappe, piu' in generale, e' necessario dare un nome solo ai campi che verranno referenziati nel programma, e quindi non ai campi costanti, di commento, di testata, ecc. Il nome e' da 1 a 7 caratteri. Nella descrizione della mappa simbolica solo i campi con un nome danno origine a "labels".
2. POS specifica la posizione relativa del byte attributo del campo all'interno della mappa, e non la posizione fisica sullo schermo. Se i dati del campo devono iniziare a linea 1 colonna 5, si indicherà

POS=(1,4)

Notiamo che la posizione da indicare e' sempre (anche per i gruppi) quella effettiva dei dati meno uno. Se la posizione e' espressa tramite coordinate, la prima posizione nella mappa e' (1,1); se espressa tramite numero relativo (displacement) la prima posizione e' indicata da 0; in questo caso l'esempio precedente POS=(1,4) si scriverebbe POS=3. La posizione dei campi e' bene che sia definita in sequenza da sinistra a destra e dall'alto in basso; e' anche possibile, pero', elencare i campi in un ordine che non rispecchi la loro posizione fisica.

3. LENGTH indica la lunghezza logica del campo (numero massimo di caratteri che esso puo' contenere)
 - La lunghezza massima e' 256
 - il byte attributo non viene conteggiato
4. ATTRB specifica le caratteristiche del campo, particolarmente in relazione a terminali 3270
 - ASKIP - autoskip (implica PROT e salto automatico del cursore al successivo campo non ASKIP)
PROT - protetto, non modificabile da tastiera
UNPROT - non protetto
NUM - numerico (per tastiera con la feature di KEYBOARD NUMERIC LOCK)
BRT - ad alta intensita' luminosa
NORM - ad intensita' normale
DRK - non visibile, non stampabile
 - IC - il cursore viene inserito all'inizio del campo. Se IC e' specificato per piu' campi, esso verrà inserito all'inizio dell'ultimo campo elencato nella mappa con ATTRB=(...,IC,...). Se per nessun campo della mappa e' stato specificato IC, il cursore viene inserito a posizione 0 dello schermo, a meno che la sua posizione non venga impostata a programma.
 - FSET imposta a 1 il bit di MDT.
MDT in on indica un campo modificato all'interno del buffer di un terminale 3270; esso verrà pertanto letto da una successiva operazione di input.
 - DET indica che il campo e' selezionabile mediante penna. Il campo deve iniziare con il carattere '?' se esso e' a selezione differita o con blank se e' a selezione immediata. La sua lunghezza deve essere almeno di cinque byte, salvo il caso in cui il campo sia l'ultimo su di una linea del video e allora puo' esser lungo due byte.
Il BMS per entrambi i campi selezionabili con penna luminosa fornisce al programma applicativo informazioni che descrivono l'attivita' dell'operatore a video. Infatti nella work-area del programma, definita dalla mappa simbolica, sono rilevanti le prime 4 posizioni per ogni campo selezionabile: il prefisso, di 3 byte, composto da 2 caratteri per la lunghezza ed un carattere riservato all'attributo (un campo selezionato dall'operatore a video puo' essere individuato a programma anche mediante il confronto con il bit MDT) mentre la quarta posizione (primo byte della parte dati del campo) conterrà un indicatore impostato a X'FF' se il campo e' stato selezionato.
5. INITIAL consente di inizializzare un campo.

La costante deve essere del tipo 'carattere', e non sono ammessi valori binari. Un campo per cui e' stato specificato INITIAL= 'costante' in generale non ha nome, salvo il caso in cui il programma non sia interessato a referenziarlo per notificarlo. Eventuali spazi presenti nella costante vanno raddoppiati.

6. JUSTIFY controlla l'allineamento di un campo in una mappa di input e l'eventuale carattere di riempimento. L'allineamento puo' essere a destra (RIGHT) o a sinistra (LEFT); i campi numerici vengono implicitamente allineati a destra ed eventualmente completati con zeri, quelli alfabetici a sinistra ed eventualmente completati con caratteri blank.
7. GRPNAM consente di combinare insieme un certo numero di campi sotto un unico nome di gruppo referenziabile a programma. Questo equivale a suddividere un campo in sottocampi. Il nome del gruppo deve essere ripetuto in ogni campo appartenente al gruppo stesso. I campi di un gruppo devono essere definiti uno di seguito all'altro; con possibilita' di spazi vuoti (quindi i campi possono essere non contigui), ma senza che ci siano campi non del gruppo.
8. OCCURS e' mutuamente esclusivo con GRPNAM. Specifica quante volte il campo deve essere ripetuto all'interno della mappa. La tabella che cosi' si crea e' accessibile mediante indice secondo le regole del COBOL.
9. PICIN/PICOUT indica che la clausola PICTURE deve essere applicata in input o in output. Il BMS verifica che il valore specificato sia conforme al linguaggio di programmazione usato. Se la lunghezza della PICTURE non e' congruente con quella indicata per il campo in LENGTH=numero, viene assunta come valida quest'ultima. Se per un campo sono stati specificati sia PICIN sia PICOUT e non sono congruenti, viene utilizzato il piu' corto dei due. Qualora non sia stato specificato LENGTH=numero, per il campo viene assunta la lunghezza della PICTURE.

3.7 CARATTERISTICHE LOGICHE DELLA MAPPA

Abbiamo già detto che contemporaneamente alla preparazione della descrizione fisica è necessaria la descrizione dei nomi simbolici con cui referenziare i campi della mappa stessa.

I parametri interessati sono tutti contenuti nella macro DFHMSD.

Analizziamo i nuovi parametri:

BASE=nome il nome e' quello di "Base Locator for Link" in COBOL. (Vedi il cap.13).

Questo parametro consente di utilizzare la stessa area di memoria per tutte le mappe elencate nel mapset, che in tal modo vengono ad essere una ridefinizione della stessa area.

STORAGE=AUTO questo parametro fa sì che ad ogni mappa del MAPSET venga riservata la propria area di memoria, ed e' mutuamente esclusivo con BASE=nome. Per semplicità di programmazione e' consigliabile usare STORAGE=AUTO, che non richiede l'uso di Base Locator for Link. Ricordiamo che se specifichiamo STORAGE=AUTO viene assunto TIOAPFX=YES anche se non e' specificato.

TIOAPFX={NO/YES} come già descritto nella macro DFHMSD, specifica se il BMS deve includere un "filler" nella definizione simbolica per gestire il prefisso della TIOA. Utilizzando i comandi del CICS/VS bisogna sempre specificare TIOAPFX=YES (il default è NO) a meno che sia assunto perché è codificato STORAGE=AUTO. Il parametro TIOAPFX può esser modificato a livello delle singole mappe codificandolo nella macro DFHMDI.

LANG={ASM/COBOL/PLI} Specifica il linguaggio dei programmi applicativi che faranno uso delle mappe di questo mapset. Se i programmi, che usano uno stesso mapset fisico, non sono scritti nello stesso linguaggio, e' necessario definire un mapset logico per ogni linguaggio.

3.8 SOMMARIO

Nella prima parte abbiamo visto le caratteristiche generali del BMS, che agisce come un interfaccia tra il programma e il TCP. Uno dei vantaggi primari del BMS e' quello di permettere la codifica di programmi senza interessarsi delle caratteristiche fisiche dei terminali.

In particolare abbiamo detto che il BMS prepara i messaggi da inviare inserendo gli appropriati caratteri di controllo per il terminale in uso.

Permette di scrivere programmi senza interessarsi della posizione fisica dei campi sullo schermo, ma basta specificare nomi simbolici di riferimento.

Il BMS si fa carico di correlare i nomi simbolici con le posizioni effettive sullo schermo facendo uso di tabelle dette "mappe fisiche" e "mappe logiche".

Una mappa fisica descrive il formato visivo dei dati per un certo tipo di terminale. Deve esserci una mappa fisica per ogni formato e per ogni tipo di terminale che ne faccia uso. Per uno stesso tipo di terminale possono esserci una o piu' mappe. Il programma usa un nome "generico" indipendente dal tipo di terminale per referenziare il mapset.

Una mappa simbolica viene usata quando un pannello contiene dati variabili che debbono essere elaborati. La funzione di una mappa simbolica e' quella di definire i campi variabili per input da (o output a) un terminale.

Nella seconda parte abbiamo visto le macro istruzioni, scritte in linguaggio ASSEMBLER, necessarie per creare i mapset fisici e simbolici. In particolare le macro:

- DFHMSD che definisce un MAPSET
- DFAMDI che definisce una MAPPA
- DFHMDF che definisce un CAMPO

Lo stesso gruppo di macro è utilizzato per creare sia il mapset fisico che il mapset logico associato.

3.9 FIGURE

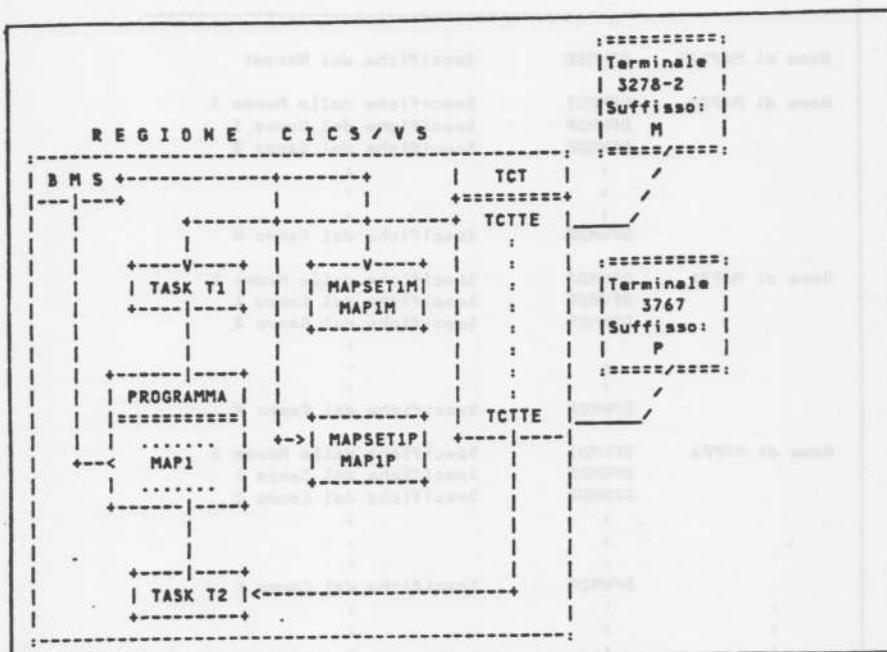


Figura 3.1 - Esempio di utilizzo di nome generico di Mapset.

STRUTTURA DI UN MAPSET

| Nome di MAPSET | DFHMSD | Specifiche del Mapset |
|-----------------------|---------------|---------------------------------|
| Nome di MAPPA | DFHMDI | Specifiche della Mappa 1 |
| | DFHMDF | Specifiche del Campo 1 |
| | DFHMDF | Specifiche del Campo 2 |
| : | : | : |
| : | : | : |
| : | DFHMDF | Specifiche del Campo N |
| Nome di MAPPA | DFHMDI | Specifiche della Mappa 2 |
| | DFHMDF | Specifiche del Campo 1 |
| | DFHMDF | Specifiche del Campo 2 |
| : | : | : |
| : | : | : |
| : | DFHMDF | Specifiche del Campo N |
| Nome di MAPPA | DFHMDI | Specifiche della Mappa 3 |
| | DFHMDF | Specifiche del Campo 1 |
| | DFHMDF | Specifiche del Campo 2 |
| : | : | : |
| : | : | : |
| : | DFHMDF | Specifiche del Campo N |
| : | : | : |
| : | : | : |
| : | DFHMSD | Chiusura del Mapset |

Figura 3.2 - Struttura di un Mapset.

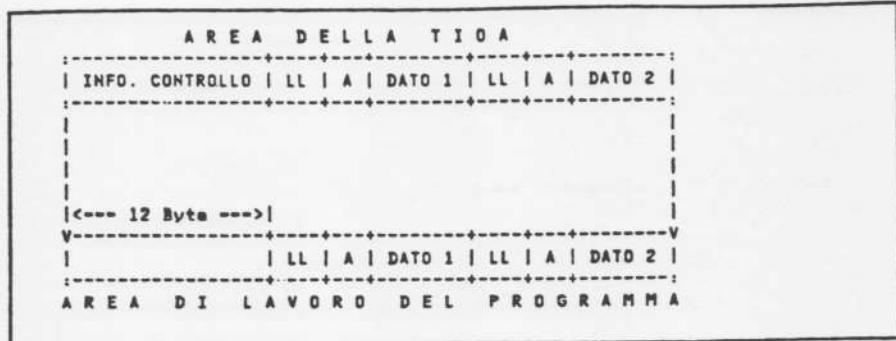


Figura 3.3 - Funzionamento del parametro TIOAPFX.

| 8 | 0 | 1 | T | 8 | 3 | 4 | 8 | 0 | 8 | 8 |
|---|---|---|---|---|---|---|---|---|---|---|
| I | S | O | T | I | A | T | I | S | O | T |
| I | C | S | T | I | A | T | I | S | O | T |
| I | C | S | T | I | A | T | I | S | O | T |
| I | C | S | T | I | A | T | I | S | O | T |

LC262 - PROGRAMMAZIONE IN AMBIENTE CICS - COBOL

4.1 INTRODUZIONE

In questo capitolo esamineremo le funzioni dell'interfaccia di programmazione "command-level" del CICS/V5 per quanto riguarda la programmazione COBOL.

4.2 LA TRANSAZIONE CICS

Una transazione CICS puo' essere attivata da uno dei terminali CICS semplicemente INTRODUCENDO IL CODICE DELLA TRANSAZIONE, lungo al piu' 4 caratteri.

Ad ogni transazione e' associato un programma, ed il CICS si interessa di gestirne l' esecuzione, cosi' come farebbe un sistema operativo, in multiprogrammazione con gli altri programmi CICS attivi in quel momento.

Cio' significa che il CICS si preoccupa di:

1. Allocare la memoria per il caricamento del programma.
2. Passare il controllo al programma per l' esecuzione delle sue istruzioni.
3. Assegnare al programma, quando questo ne faccia richiesta, le risorse CICS che sono condivise da tutti gli utenti, cioè file, record di Data Base, spazio su disco e in memoria.

Un comando CICS particolare, il comando RETURN, fa terminare la transazione CICS.

4.3 I SERVIZI DEL CICS

Nei programmi legati alle transazioni CICS e' quindi spesso necessario far richiesta al CICS per ottenere una delle risorse del sistema.

Si possono distinguere le richieste di risorse a seconda della RISORSA CICS interessata :

1. richiesta di trasmissione e ricezione di messaggi da terminal (BMS)
2. richiesta di accesso ad un file (File Control)

3. richiesta di accesso ad un segmento di un Data Base DLI (CICS/DLI Interface)
4. richiesta di accodamento di dati temporanei (Transient Data o Temporary Storage)
5. richiesta di passaggio del controllo e scambio di dati fra programmi (Program Control)
6. richiesta di terminazione di una transazione (Program Control)
7. richiesta di memoria centrale (Storage Control)
8. richiesta di temporizzazione dell' esecuzione dei programmi (Interval Control)

Abbiamo scritto vicino ad ogni richiesta il MODULO CICS che e' responsabile di gestire la richiesta corrispondente.

E' proprio attraverso istruzioni particolari, COMANDI CICS, che il programmatore richiede l' intervento del modulo CICS che offre il servizio richiesto.

Nel far cio' il modulo CICS segue le indicazioni fornite dal programmatore come OZIONI del comando.

Terminato il servizio, il controllo ritorna al programma, ed esattamente alla istruzione successiva al comando CICS.

6.4 L' INTERFACCIA CICS/VS - COMANDI

Saper scrivere programmi in ambiente CICS significa essere in grado di inserire in un programma COBOL i COMANDI CICS necessari.

Il linguaggio dei comandi CICS e' una specie di ESTENSIONE del linguaggio COBOL, e infatti si dice che il COBOL fa' da linguaggio OSPITE per i comandi CICS.

Attraverso questi comandi, come abbiamo detto, si possono richiedere al CICS servizi di accesso ed utilizzo delle risorse del sistema (ad esempio leggere un file).

I comandi sono simili ad istruzioni COBOL, e possono essere inseriti in un programma applicativo ovunque sia necessario.

Pertanto un programma COBOL-CICS puo' contenere sia istruzioni COBOL sia comandi CICS/VS.

4.4.1 IL FORMATO DEI COMANDI

Il formato generico di un comando CICS/VS e':

EXECUTE CICS
oppure abbreviato:
EXEC CICS

seguito dal nome della funzione desiderata e da una o piu'opzioni; cioè'

EXEC CICS funzione opzione (argomento)
opzione (argomento).....

Ogni comando CICS/VIS deve essere sempre chiuso con un delimitatore particolare, che in COBOL e' END-EXEC.

La parola chiave "funzione" specifica l'operazione richiesta. Per esempio READ richiede la lettura di un file.

Ad ogni comando EXEC CICS puo' essere associata una sola funzione.

La parola chiave "opzione" descrive una delle possibili specifiche, associate con una particolare "funzione".

Alcune opzioni prevedono un proprio "argomento", altre no. Tutte, comunque, possono essere scritte in un ordine qualsiasi.

Un esempio di codifica di un comando di lettura puo' essere:

```
EXEC CICS READ  
FILE ('FILEA') GTEQ RIDFLD(CHIAVE)  
INTO(AREA) END-EXEC
```

dove:

- FILE indica il file su cui leggere;
- RIDFLD indica il campo che contiene la chiave di ricerca;
- GTEQ indica il tipo di ricerca (per maggiore o uguale del campo CHIAVE);
- INTO indica l'area di lavoro che deve ricevere il record.

4.4.2 I VALORI DEGLI ARGOMENTI

Il valore di un argomento puo' essere scritto in due modi.

a) Se e' definito all'interno del programma, esso e' chiuso in parentesi, ma non vuole apici. Esempio: RIDFLD(CHIAVE) o INTO(AREA)

b) Se non e' definito nel programma, ma e' noto al CICS/V5, allora e' chiuso in parentesi e vuole gli apici. Esempio: FILE('FILEA')

Un argomento puo' assumere uno dei seguenti valori:

- Data-value
- Data-area
- Pointer-value
- Pointer-reference
- Name
- Label
- Time (hhmmss)

In COBOL questi valori significano:

Data-value Nome di costante o di campo avente una struttura del tipo:

- mezza voce binaria: PIC S9(4) COMP
- una voce binaria: PIC S9(8) COMP
- stringa di caratteri: PIC X(n)

Data-area Nome di campo rappresentante una struttura del tipo:

- mezza voce binaria: PIC S9(4) COMP
- una voce binaria: PIC S9(8) COMP
- stringa di caratteri: PIC X(n)

Pointer-value Nome di elemento di BLL (Base Locator for Linkage), oppure nome di campo che contenga una copia di un elemento di BLL (vedi cap. 13).

Pointer-ref Nome di un elemento di BLL. (vedi cap.13).

Name Costante non numerica tra apici, oppure nome di un campo lungo quanto il massimo della costante. Il valore nel campo e' il nome utilizzato come argomento.

Label Nome di paragrafo o di sezione Cobol.

hhmmss Costante decimale packed, oppure nome di campo di forma
PIC 59(7) COMP-3.

La spiegazione dettagliata del significato dei suddetti valori è reperibile al Capitolo 1.2 del Manuale del Programmatore CICS/V5 - Comandi (il cui titolo originale è: CICS/V5 - Application Programmer's Reference Manual SC33-0512, indicato in seguito con la sigla APRM).

4.4.3 IL TRADUTTORE DEI COMANDI

Poiché il compilatore COBOL non è in grado di interpretare i comandi del CICS/V5, è necessario che questi siano opportunamente tradotti, prima di compilare il programma che li contiene.

Per questa esigenza è disponibile un programma chiamato "Traduttore dei Comandi" (Command Translator).

La funzione di questo programma traduttore è di accettare in input il programma Cobol, contenente anche comandi CICS/V5, e di produrre in output un programma equivalente, in cui ogni comando CICS/V5 è tradotto in una o più istruzioni MOVE, seguite da una CALL parametrica. Il proposito delle istruzioni MOVE è di assegnare delle costanti ai dati variabili del programma Cobol; permettendo così di specificare costanti e nomi come "argomenti" di "opzioni" nei comandi.

Durante l'esecuzione, queste istruzioni inserite dal traduttore invocano l'EXEC INTERFACE PROGRAM (DFHEIP), che accetta gli argomenti passati dal programma attraverso i parametri della CALL, imposta i parametri nei blocchi di controllo del CICS/V5 e, infine, cede il controllo al modulo del CICS/V5 richiesto nel comando.

Prima di poter eseguire un programma applicativo Cobol, contenente comandi CICS/V5, sono necessari tre passi operativi:

1. TRADUZIONE
2. COMPILAZIONE
3. CATALOGAZIONE

Il traduttore legge i dati in input da SYSIN, produce l'output (cioè il programma sorgente tradotto) su SYSPUNCH e scrive la lista ed i messaggi di errore su SYSPRINT.

4.4.4 LE OPZIONI PER IL TRADUTTORE

E' possibile fornire al traduttore dei comandi CICS/V5 un certo numero di opzioni, inserendo, in testa al programma, la scheda CBL seguita dalla parola chiave XOPTS.

La scheda CBL puo' contenere anche opzioni per il compilatore, ignorate peraltro dal traduttore, che le passera' invariate al compilatore.

Invece della scheda CBL, le opzioni possono essere specificate anche nella proposizione EXEC di Job Control che invoca il traduttore.

Se sono usati entrambi i metodi, prevale quanto specificato nella proposizione EXEC.

Nel Capitolo 1.3 dell'APRM e' possibile reperire la descrizione completa di tutte le possibili opzioni del traduttore.

4.4.5 RESTRIZIONI DEL COMPILATORE COBOL

Alcune istruzioni COBOL che hanno senso in ambiente BATCH diventano priva di significato in contesto CICS, in quanto lo stesso servizio e' ottenibile mediante un comando CICS invece che attraverso una istruzione COBOL.

Per questo motivo il manuale di riferimento porta una lista di istruzioni COBOL che non vengono piu' riconosciute dal compilatore COBOL sotto CICS.

Non e' necessario riportare la lista in questo testo, perch'e' conosciuti i comandi CICS, non nasce neanche l'esigenza di usare l'istruzione COBOL corrispondente.

Nel Capitolo 1.4 dell'APRM e' possibile reperire l'elenco completo delle restrizioni cui e' soggetto un programma Cobol in ambiente CICS/VS, oltre alle informazioni sull'uso degli elementi di BLL (Base Locator for Linkage; vedi anche cap.13).

Per dettagli ulteriori sulla programmazione Cobol-CICS/VS e' bene consultare l'apposita Guida del Programmatore COBOL.

4.5 IL BLOCCO EIB (EXECUTE INTERFACE BLOCK)

In aggiunta ai servizi eseguiti dai vari moduli di gestione (come la lettura di un record), il CICS/VS mantiene automaticamente certe informazioni (come il numero del task, la data, l'ora,...) che potrebbero interessare anche il programma applicativo durante la sua esecuzione.

Tali informazioni, normalmente contenute in aree non accessibili al programma, vengono automaticamente copiate dal CICS/VS nel blocco EIB (EXECUTE INTERFACE BLOCK) creato nel momento in cui inizia il task.

Esiste quindi un blocco EIB per ogni task e il traduttore dei comandi ne include automaticamente una copia nella Linkage Section di ogni programma Cobol.

I nomi dei campi dell'EIB sono nomi riservati e il programma applicativo puo' solo leggerne il contenuto.

Per esempio sara' sufficiente un'istruzione del tipo:

- MOVE EIBxxx TO AREAEB

per rintracciare l'informazione contenuta nel campo EIBxxx e trasferirla nel campo utente AREAEB.

Nell'Appendice A dell'APRM sono contenuti tutti i codici EIB con la loro descrizione dettagliata ed il loro utilizzo.

4.6 LE CONDIZIONI ANOMALE

Un qualunque servizio richiesto al CICS, termina in uno dei due modi mutuamente esclusivi :

1. SERVIZIO TERMINATO CORRETTAMENTE : non ci sono stati problemi durante l'esecuzione del comando CICS ed il programma puo' continuare regolarmente l'elaborazione.
2. SERVIZIO TERMINATO IN MODO ANOMALO : durante l'esecuzione del comando CICS si e' verificata una condizione eccezionale che ha impedito che il servizio si svolgesse nel modo richiesto. Il programma non puo' proseguire regolarmente ma deve far fronte alla nuova situazione.

Ad esempio, in seguito di una richiesta di READ di un certo record da un file VSAM, di cui abbiamo specificato la chiave tra le opzioni del comando, potrebbe darsi che il record non venga trovato.

In questo caso il File Control Program, che e' il responsabile della lettura di record su file, comunica al programma che e' stata registrata la condizione eccezionale di RECORD NON TROVATO.

Il verificarsi di una condizione eccezionale (detta anche condizione anomala) non e' sempre un fatto della medesima gravita': ogni comando comporta, per sua natura, possibilita' di differenti malfunzioni.

Percio' e' utile che il programma applicativo disponga di strumenti atti a gestire le diverse condizioni anomale in modo diversificato, cosi' da poter intraprendere azioni diverse a fronte di circostanze anomale diverse.

Il programmatore deve attivare la gestione delle condizioni anomale prima che queste si verifichino. Per alcune di queste il programmatore puo' inserire nel programma le istruzioni necessarie per gestirle nel momento in cui si verificano; per altre puo' accettare le azioni standard previste dal CICS/VS.

Per la gestione delle condizioni eccezionali esistono due modalità diverse: i comandi chiamati HANDLE e IGNORE, che permettono di specificare l'azione da intraprendere in ogni situazione anomala prima di schedulare un comando e le opzioni NOHANDLE e RESP all'interno del comando.

Se il programmatore non specifica cosa dovrebbe essere fatto. (includendo il comando HANDLE, un comando IGNORE o una opzione NOHANDLE o RESP) il CICS/VIS prevederà una gestione standard che risulterà in un ABEND del task.

Se, al verificarsi di una certa condizione eccezionale, non deve essere intrapresa nessuna azione si può codificare nel programma il comando IGNORE per quella condizione o l'opzione NOHANDLE nel comando che genera la condizione eccezionale.

Il formato del comando HANDLE CONDITION è :

```
EXEC CICS HANDLE CONDITION  
    condizione ((label))  
    (condizione((label)) ...)
```

La condizione è l' IDENTIFICAZIONE DELL'ERRORE (per esempio record duplicato, richiesta invalida, etc ...) e la label identifica la particolare routine cui si deve trasferire il controllo quando si verifica quella particolare condizione.

In ogni singolo comando di tipo HANDLE CONDITION si possono specificare più condizioni anomale, fino ad un massimo di dodici.

E' bene averle presenti in fase di codifica del comando, per stabilire quali tra quelle descritte, si intende gestire nel programma.

Nel capitolo 1.5 dell'APRM si trova una 'Lista delle Condizioni Anomale' associate al comando che può generarle.

Nel seguito del testo, tutte le volte che si parla di un comando si citano anche le principali condizioni anomale che questo comando prevede.

Per disattivare la gestione di una condizione eccezionale, ed intraprendere una azione correttiva implicita (default), si può codificare un comando EXEC CICS HANDLE CONDITION senza specificare la label per la 'condizione' specificata o codificare un comando EXEC CICS IGNORE per la 'condizione'.

Come esempio vediamo il caso di una scrittura di un record su un file.

Una condizione possibile (e neanche infrequente) è che esista già sul file un record con lo stesso identificatore; si tratta di una condizione che si può gestire con il comando :

```
EXEC CICS HANDLE CONDITION DUPREC(RTHA)
      ERROR(STOP)
      END-EXEC
```

Si noti che e' possibile codificare il comando su una stessa riga di programma; il nome della condizione e' DUPREC, che significa 'RECORD DUPLICATO'.

Il nome della routine cui cedere il controllo e' RTHA; in particolare esso rappresenta il nome del paragrafo a cui verrà ceduto il controllo ogni volta che il programma si troverà in presenza di record doppi.

Nell'esempio compare anche la condizione ERROR. Questa e' una condizione eccezionale di tipo generico, che passa il controllo al paragrafo STOP ogniqualvolta si verifica una condizione eccezionale non specificata esplicitamente in un comando HANDLE CONDITION.

Lo scopo e' evidentemente quello di poter gestire TUTTE le condizioni eccezionali che non si puo' o vuole prevedere minuziosamente.

Il comando HANDLE CONDITION e' di solito codificato all'inizio del programma in quanto, una volta specificata, la condizione resta VALIDA fino a:

1. alla chiusura del programma
2. all'esecuzione di un altro comando HANDLE per la stessa condizione
3. all'esecuzione di un comando IGNORE per la medesima condizione.

Se peraltro in un qualche punto del programma e' richiesta un'azione diversa a fronte di una stessa condizione eccezionale, occorrerà codificare un NUOVO COMANDO HANDLE CONDITION che specifichi la nuova azione richiesta da quel punto in poi e automaticamente cancella l'influenza del comando precedente.

Ad esempio, volendo in seguito gestire in modo diverso la condizione eccezionale DUPREC, basterà codificare un'altra HANDLE CONDITION DUPREC(label), dove il valore di label sara' diverso da RTHA.

6.6.1 Comando IGNORE

Il comando IGNORE specifica che non deve essere intrapresa nessuna azione quando si verifica una certa condizione eccezionale. Il formato del comando e':

EXEC CICS IGNORE CONDITION condizione condizioni ...

Condizione specifica il nome della condizione eccezionale da ignorare.

Se, per esempio, non deve essere intrapresa nessuna azione quando in un file non viene trovato un certo record il comando deve essere codificato come segue :

EXEC CICS IGNORE CONDITION NOTFND

In un comando IGNORE si possono specificare al massimo dodici condizioni.

4.6.2 Opzione NOHANDLE

L' opzione NOHANDLE puo' essere codificata in ogni comando per specificare che non deve essere intrapresa alcuna azione per TUTTE le condizioni anomale risultanti dall' esecuzione del comando.

Per esempio l'opzione NOHANDLE codificata nel seguente comando :

```
EXEC CICS READ DATASET('MASTER')
      RIDFLD(CHIAVE)
      INTO(AREA)
      NOHANDLE
      ....
```

sopprime tutte le condizioni eccezionali del comando READ.

Ricordare che per la gestione delle condizioni eccezionali esistono 4 alternative che possono essere usate singolarmente o in combinazione :

- Scrivere una routine specifica, come DUPREC o NOTFND, per la gestione della specifica condizione.
- Scrivere una routine generalizzata, come ERROR, per gestire tutte (o parte) le condizioni eccezionali.

Notare che tale routine deve poter differenziare la gestione delle possibili cause di errore.

- Codificare un comando IGNORE per ignorare un specifica condizione.

- Lasciare la gestione a carico del CICS.

In ogni caso i comandi HANDLE CONDITION e IGNORE CONDITION devono essere codificati prima dell'esecuzione dei comandi che possono generare le condizioni eccezionali da gestire.

Per quelle condizioni poi che possono verificarsi a seguito di comandi diversi (per esempio NOTFND potrebbe verificarsi sia a seguito di un comando di FILE CONTROL che a seguito di un comando di Temporary Storage) può essere utile conoscere quale funzione ha in quel momento originato l'anomalia.

Questo può esser fatto esaminando il campo EIBFN del blocco EIB, che contiene il codice del comando CICS appena eseguito (vedi appendice A dell'APRM).

4.6.3 Opzione RESP

L'opzione RESP può essere specificata all'interno di ciascun comando CICS per specificare che non deve essere intrapresa nessuna azione per tutte le condizioni che si possono verificare durante l'esecuzione del comando stesso.

Specificare l'opzione RESP equivale quindi a NOHANDLE per quanto riguarda la gestione dell'esecuzione del comando senza però dover rinunciare ad ottenere l'informazione sull'esito del comando stesso.

La sintassi del comando è la seguente:

```
EXEC CICS .....  
.....  
RESP (NRESP)  
END-EXEC.
```

dove NRESP è una variabile definita all'interno del programma applicativo (specifico COBOL PIC S9(9) COMP) che CICS valorizzerà in modo opportuno a seconda dell'esito del comando.

La verifica del contenuto della variabile NRESP può essere fatto in due modi:

a) confrontando il contenuto della variabile con i valori corrispondenti alle condizioni da controllare, esempio:

```
IF NRESP = 36  
THEN  
.....gestione del mapfail
```

b) confrontando il contenuto della variabile con il risultato di una macro istruzione CICS

IF NRESP = DFHRESP(MAPFAIL)
THEN
.....gestione del mapfail

Gli esempi proposti controllano entrambi il verificarsi della condizione di nessun dato digitato con la differenza che nel secondo caso si fa riferimento al nome simbolico della condizione mentre nel primo si deve conoscere il valore esatto attribuito da CICS alla variabile NRESP.

L'utilizzo dell'opzione RESP nei comandi E' FORTEMENTE CONSIGLIATO in quanto consente una maggior flessibilità rispetto all'uso della HANDLE CONDITION e soprattutto consente una maggiore chiarezza nella stesura del programma applicativo evitando i salti incondizionati impliciti nella HANDLE CONDITION stessa.

Per maggiori dettagli riguardo l'uso dell'opzione RESP vedi anche il 1.5 dell'APRM.

4.7 SOMMARIO

In questo capitolo abbiamo visto come il CICS agisce da sistema operativo per le risorse sotto il suo controllo.

Il programmatore deve perciò richiedere al CICS quei servizi che in ambiente batch sono controllati dal sistema operativo, come per esempio l'accesso ai files, al Data Base o la schedulazione di un programma.

Per ottenere questi servizi dal CICS il programmatore deve introdurre nei suoi programmi COBOL i comandi CICS adeguati.

Questi comandi sono tradotti in CALL a moduli del CICS, in cui i parametri di chiamata sono presi dall'elenco di opzioni previste per quel comando.

Per ciascun diverso servizio richiesto attraverso un comando, sono previste tutte le possibili condizioni eccezionali con cui il servizio puo' terminare.

E' compito del programmatore scrivere le routine di gestione di quelle condizioni che vuole tenere sotto controllo, e scrivere il nome di tali routine (o paragrafi), come opzione del comando di HANDLE CONDITION appropriato.

5.1 INTRODUZIONE

Nella prima parte di questo capitolo vedremo quali sono i comandi BMS e come essi utilizzano le mappe per comunicare con i terminali (I/O MAPPING).

Nella seconda parte, invece, vedremo brevemente i comandi CICS/VS relativi alle funzioni del Terminal Control Program (TCP), che e' l'effettivo responsabile delle comunicazioni tra i programmi applicativi ed i terminali della rete.

PARTE PRIMA

5.2 I/O MAPPING

Nel capitolo 3 abbiamo visto come e' possibile creare le mappe fisiche e (quelle) simboliche necessarie per rendere accettabile un messaggio proveniente da un terminale o per dare un formato ai dati ad esso indirizzati.

In questo capitolo impareremo a codificare i comandi BMS che utilizzano le mappe per comunicare con i terminali (I/O MAPPING).

Sappiamo che (paragrafo 3.4.1), durante l'assemblaggio delle mappe logiche, i nomi dei campi in esse definiti subiscono delle trasformazioni. In particolare, nelle mappe logiche di INPUT, i nomi ricevono il suffisso I, in quelle di OUTPUT il suffisso O.

Per esempio, a fronte di un Mapset logico del tipo:

```

MAPSETA DFHMSD TYPE=DSECT.
      LANG=COBOL.
      TIOAPFX=YES.
      STORAGE=AUTO.
      MODE=INOUT
MAPA ... DFHMFI SIZE=.....
      DFHMDF POS=.....
      LENGTH=.....
      ATTRB=.....
      -----
      NAME    DFHMDF POS=.....
              LENGTH=20.
              ATTRB=.....
      -----
      DEPT   DFHMDF POS=.....
              LENGTH=12.
              ATTRB=.....

```

```
.....  
DFHMSD TYPE=FINAL  
END
```

le istruzioni generate durante l'assemblaggio saranno:

```
01 MAPAI.  
02 FILLER PIC X (12).  
02 NAMEL PIC S9 (4) COMP.      "LUNGHEZZA"  
02 NAMEA PIC X.              "ATTRIBUTO"  
02 FILLER REDEFINES NAMEA.  
03 NAMEF PIC X.              "FLAG"  
02 NAMEI PIC X (20).          "NOME-INPUT"  
02 DEPTL PIC S9 (4) COMP.  
02 DEPTA PIC X.  
02 FILLER REDEFINES DEPTA.  
03 DEPTF PIC X.  
02 DEPTI PIC X (12).  
-----  
01 MAPAO REDEFINES MAPAI.  
02 FILLER PIC X (12).  
02 FILLER PIC X (3).  
02 NAMEO PIC X (20).          "NOME-OUTPUT"  
02 FILLER PIC X (3).  
02 DEPTO PIC X (12).  
-----
```

Le precedenti definizioni, generate dall'assemblaggio delle mappe logiche, devono essere copiate all'interno del programma per poter essere utilizzate. Esse, quindi, vengono prima catalogate in una libreria "source", sotto un certo nome scelto dal programmatore di sistema e comunicato al programmatore applicativo.

Supponiamo che il nome di catalogazione in "source" sia quello della macro DFHMSD TYPE=DSECT, cioè, nel nostro esempio, MAPSETA.

In MAPSETA la prima macro di definizione di mappa ha il nome MAPA. Percio' un programmatore COBOL, per copiare la DSECT corrispondente al mapset MAPSETA, deve codificare nella WORKING-STORAGE SECTION:

```
COPY MAPSETA
```

cioè' il nome della prima mappa del mapset deve comparire a livello 01 con il suffisso I.

5.3 INPUT MAPPING

Quando inviamo un comando di ricezione di una mappa, il TCP legge i dati dal buffer del terminale e li muove nell'Area di I/O del Terminale, (TIOA).

Il messaggio, cosi' come lo riceve il TCP, e' una stringa di dati che si presenta in "modo nativo". Essa e' composta da:

- I dati modificati immessi a terminale dall'operatore o definiti nella mappa come "modificati" (ATTRB=FSET);
- I caratteri di controllo, che sono intercalati con i dati e specifici per il terminale.

Il BMS rimuove i caratteri di controllo e mette i dati in un buffer. Già in questo buffer i dati potrebbero essere elaborati, ma solitamente essi vengono mossi nella appropriata area di lavoro del programma (vedi paragrafo precedente) ed acceduti usando i nomi suffissati dei campi della mappa logica di INPUT.

Ricordiamo che i suffissi dei campi indirizzabili in una mappa di input sono:

L per il campo lunghezza (una mezza voce binaria);

F per il "flag" di un byte;

I per il nome del campo dati.

Quindi il campo dati (nome-I) e' preceduto da due bytes del campo lunghezza (nome-L) e da un byte del campo "flag" (nome-F).

La lunghezza contenuta nel campo nome-L e' quella reale del dato trasmesso e non puo' eccedere mai la lunghezza massima specificata nel campo della mappa fisica (parametro LENGTH).

Per controllare a programma se l'operatore abbia immesso dati in un campo, e' possibile testare il campo lunghezza (nome-L). Se questo contiene zero, non e' stato digitato alcun dato. In particolare, se il campo "flag" (nome-F) contiene X'80', e' stato premuto il tasto di Fine Campo (EOF), senza peraltro introdurre dati.

Se il campo nome-L contiene, invece, un valore diverso da zero, verranno trasmessi i dati digitati e la loro lunghezza.

5.3.1 Il comando RECEIVE MAP

Il comando RECEIVE MAP e' quello che consente di trasmettere il contenuto di tutti i campi modificati di una mappa, da un terminale al programma applicativo richiedente.

Dopo l'esecuzione del comando, il BMS pone tutte le informazioni sui dati modificati nell'area di lavoro definita appropriatamente dal programma, usando il formato e i nomi specificati nella mappa fisica e simbolica.

Il comando BMS completo e':

EXEC CICS RECEIVE MAP (nome)

```
MAPSET (nome)
  (INTO (area-dati)/SET (puntatore))
  (FROM (area-dati)LENGTH (numero)/
  TERMINAL(ASIS))
```

Nella sua forma comune e' sufficiente specificare il nome della mappa e quello del mapset che la contiene, entrambi tra apici. Se il mapset ha lo stesso nome della mappa, non e' necessario specificare nemmeno il mapset. In questo modo il BMS colloca i dati di input nella struttura dati della mappa simbolica.

Quindi la forma piu' semplice del comando RECEIVE MAP contiene costanti letterali per il mapset e la mappa. Talvolta, pero', potrebbe essere utile usare nomi variabili: per questo sono disponibili le opzioni INTO e SET.

INTO permette di ottenere i dati modificati dalla mappa in un'area dati definita nel programma, ma diversa dalla struttura dati della mappa simbolica.

SET, invece, richiede al BMS di acquisire un'area dati esterna al programma, di collocarci i dati in input, e di copiarne l'indirizzo nel puntatore indicato da SET (vedi Cap.13).

Per default, viene assunto che i dati in input da una mappa provengano da un TERMINALE.

In particolare, l'opzione ASIS indica che non si vuole la traduzione dei dati da minuscolo a maiuscolo.

Talvolta ci puo' essere la necessita' di eseguire dell'"input mapping" in due passi: accettazione e memorizzazione di dati, nel primo, "mapping" di questi nel secondo.

Per esempio, se i dati di una mappa sono ricevuti in un'area di lavoro del programma da un comando RECEIVE di Terminal Control (vedi piu' avanti la Seconda Parte del capitolo), sicuramente essi non hanno subito il "mapping", specifico nel BMS. Sarà poi possibile, usando il comando RECEIVE MAP con le opzioni FROM e LENGTH, ottenere i dati di input dall'area di lavoro che li contiene, invece che da un terminale. FROM indica il nome dell'area dei dati e LENGTH la lunghezza associata.

Dopo un comando RECEIVE MAP, e' possibile conoscere la posizione del cursore nella mappa di input (EIBCPOSH) e il tipo di tasto funzionale, o Attention Identifier (AID), utilizzato per inviare la mappa (EIBAID).

Notiamo che, dopo un comando RECEIVE MAP, la condizione di MAPFAIL puo' verificarsi inaspettatamente.

Infatti, perche' questo accada, e' sufficiente premere il tasto CLEAR (o un qualsiasi tasto PA) oppure il tasto ENTER (o un qualsiasi tasto PFx) senza immettere dati, quando il CICS e' in attesa di elaborare un comando RECEIVE.

Percio' sara' sempre opportuno codificare un comando di HANDLE CONDITION per la condizione di MAPFAIL.

5.4 OUTPUT MAPPING

Ogni volta che un programma applicativo desideri comunicare dei dati ad un terminale, utilizzando delle mappe, e' necessario inviare un comando di "output mapping".

A seguito di tale richiesta, il BMS inserisce gli appropriati caratteri di controllo all'inizio di ogni campo dati, generando una stringa dati, che viene collocata nella TIOA e, poi, trasmessa al terminale dal TCP.

Ricordiamo che suffissi dei campi indirizzabili e modificabili di una mappa di OUTPUT, sono:

A per il byte attributo definente le caratteristiche del campo dati da inviare (per esempio protetto o/e luminoso);

O per ogni campo dati da spedire al terminale.

Per mappe simboliche generate con l'opzione di "attributi estesi" (EXTATT=YES nelle macro DFHMDI o DFHMSD), saranno disponibili anche i suffissi H, P, V e C per i quali si rimanda all'APRM Cap. 3.2.2.

Vedremo che risultera' utile la possibilita' di modificare temporaneamente a programma gli attributi dei campi presenti nelle mappe prima di inviarle a terminale.

5.4.1 Il Comando SEND MAP

Il comando SEND MAP e' lo strumento fornito dal BMS per fare dell'OUTPUT MAPPING. Esso permette di trasmettere una mappa a terminale, (quindi dati costanti e variabili con i relativi attributi) e di controllare le modalita' di output.

Nella sua forma piu' semplice si codifica:

```
EXEC CICS SEND MAP (nome di mappa)
  (MAPSET (nome di mapset))
  (TERMINAL)
  (FROM (nome di area) LENGTH (numero))
  (DATAONLY/MAPONLY)
  (controllo.....)
```

MAP (nome) e MAPSET (nome) individuano la mappa ed il mapset da utilizzare nell'esecuzione del comando. Se il mapset ha lo stesso nome della mappa, non e' necessario specificarlo.

TERMINAL

Indica che i dati devono essere inviati al terminale dopo l'editazione. E' assunto implicitamente.

FROM (nome di area)

Indica l'area entro cui il programma ha preparato i dati da inviare al terminale. Se e' omesso, il BMS assume che l'area contenente i dati sia quella definita dalla mappa simbolica copiata all'interno del programma il cui nome compare in MAP (nome). Incompatibile con MAPONLY.

LENGTH (numero)

Indica la lunghezza dell'area contenente i dati.

Se si usa FROM, l'area contenente i dati deve avere un prefisso di 12 byte. Questo prefisso per le mappe e' automaticamente assicurato dal parametro TIDAPFX=YES.

DATAONLY

Richiede l'invio al terminale dei soli dati preparati dal programma, senza eventuali costanti contenute nella mappa fisica.

MAPONLY

Richiede l'invio dei soli dati costanti contenuti nella mappa. I dati preparati dal programma applicativo vengono ignorati. Viene utilizzato quando si desidera eseguire l'operazione di "formattazione dello schermo", cioè quando si vuol far comparire sullo schermo del terminale una maschera che servira' da guida all'operatore per l'introduzione dell'input. Ovviamente, trattandosi di intestazioni, costanti, commenti, essi sono completamente definiti nella mappa.

Se nulla e' specificato, verranno inviati al terminale sia i dati preparati dal programma, sia le costanti contenute nella mappa fisica.

Alcuni parametri codificabili nel comando SEND, utili per il 3270, servono a modificare temporaneamente le modalita' di esecuzione dell'output indicate nella mappa fisica. Essi sono indicati col nome generico "controllo":

ALARM

Aziona l'allarme acustico.

CURSOR ((numero))

Posiziona il cursore sullo schermo all'indirizzo indicato nell'argomento oppure in modo "dinamico" (senza argomento). Vedi il paragrafo successivo.

ERASE/ERASEAUP

Cancellazione di tutto lo schermo (ERASE) o solo del contenuto dei campi non protetti (ERASEAUP).

FREEKB

Sblocca la tastiera.

FRSET

Imposta a zero gli MDT.

HONEOM/L40/L64/L80

Individua la lunghezza della linea di stampa per i stampatrici del sistema 3270, oppure fa sì che i caratteri NL e EOM presenti nel messaggio (prennotati dal programma) vengano onorati.

NLEOM

Richiede al BMS la costruzione di un messaggio con suddivisione in linee tramite NL e segnalazione della fine dei dati tramite EOM. Applicabile solo per stampatrici del sistema 3270.

PRINT

Richiede l'inizio delle operazioni di stampa.

FORMFEED

Richiede un salto pagina. Utile per stampare delle schermate su pagine nuove.

Nella sua forma più semplice il comando di SEND MAP contiene solo il nome della mappa fisica di output e, se differente, quello del mapset che lo contiene.

Prima di inviare il comando SEND MAP il programma assegna dei valori utente alle variabili descritte nella mappa simbolica. Eseguendo il comando, il BMS sostituisce i dati applicativi ai dati e agli attributi di default della mappa fisica, quindi trasmette la mappa modificata al terminale.

Poiché il BMS usa sempre i dati applicativi e gli attributi presenti nella struttura della mappa logica (invece che i valori corrispondenti presenti nella mappa fisica) è bene iniziare un programma impostando a zeri binari la mappa logica di output.

5.4.2 Posizionamento del Cursore

Se l'opzione CURSOR non viene specificata, il BMS colloca il cursore all'inizio del campo della mappa definito con l'attributo IC (parametro ATTRB=IC nella macro DFHMDF).

Abbiamo appena visto, però, che è possibile modificare la posizione di default del cursore in due modi:

1. opzione CURSOR con argomento;

2. opzione CURSOR senza argomento.

Nel primo caso, per esempio, possiamo scrivere

CURSOR (10)

ottenendo il posizionamento del cursore nella posizione numero 9 dello schermo (la prima, infatti, ha indirizzo zero).

Nel secondo caso possiamo far uso del Posizionamento Simbolico del Cursore (SCP). Per far questo, dobbiamo:

1. specificare una mappa di I/O (MODE=INOUT nella macro DFHMSD);
2. impostare a -1 la lunghezza del campo sotto cui deve posizionarsi il cursore;
3. eseguire il comando SEND MAP, specificando CURSOR senza argomento.

5.4.3 Modifica Dinamica dell'Attributo

Il byte attributo di ogni campo e' definito nella mappa simbolica (nome-A) e puo' essere modificato dal programma applicativo prima di inviarlo a terminale.

Poiche' le particolari combinazioni di bit rappresentanti gli attributi sono difficili da ricordare, il CICS/VS fornisce una lista di nomi standard per gli attributi, che e' possibile utilizzare a programma invece delle corrispondenti configurazioni di bit.

Questa lista di dichiarative e' catalogata come DFHBMSCA e per includerla nel corpo di un programma COBOL bisogna codificare un'istruzione di COPY nella WORKING-STORAGE SECTION.

Tale lista di dichiarative semplifica notevolmente l'uso dell'attributo e del carattere di controllo per terminali stampanti.

I nomi simbolici contenuti in DFHBMSCA sono evidenziati qui di seguito.

| NOME SIMBOLICO | SPECIFICHE PER IL CARATTERE DI CONTROLLO (STAMPANTI) E PER IL CAMPO ATTRIBUTO |
|----------------|---|
| DFHBMASB | AUTOSKIP ALTA INTENSITA' |
| DFHBMAZF | AUTOSKIP MDT IN ON |
| DFHBMASK | AUTOSKIP |
| DFHBMBRY | ALTA INTENSITA' LUMINOSA |
| DFHBMDAR | NON VISIBILE/STAMPABILE |
| DFHBMFSE | MDT IN ON |
| DFHBMPFM | FINE MESSAGGIO PER PRINTER 3270 |
| DFHBMPNL | NEW-LINE PER PRINTER 3270 |
| DFHBMPRF | PROTETTO E MDT IN ON |
| DFHBMPRO | PROTETTO |

| | |
|----------|---|
| DFHBMUHN | NON PROTETTO E NUMERICO |
| DFHBMUNP | NON PROTETTO |
| DFHBLUE | BLU |
| DFHRED | ROSSO |
| DFHPINK | ROSA |
| DFHGREEN | VERDE |
| DFHTURQ | TURCHESE |
| DFHYELLO | GIALLO |
| DFHNEUTR | NEUTRO |
| DFHBLINK | LAMPEGGIANTE |
| DFHREVR5 | CARATTERE IN NEGATIVO |
| DFHUNDLN | SOTTOLINEATO |
| DFHMFIL | IL CAMPO DEVE ESSERE RIEMPIUTO |
| DFHMENT | DATI OBBLIGATORI NEL CAMPO |
| DFHIME | RIEMPIMENTO OBBLIGATORIO E DATI OBBLIGATORI |

Per un elenco completo vedi APRM cap. 3.2-5

5.5 LE CONDIZIONI ANOMALE

Sappiamo che quando il CICS/VIS non e' in grado di eseguire esattamente un comando come emesso dal programma, si verifica una condizione anomala.

Abbiamo visto nei capitoli precedenti che il comando HANDLE identifica sia le condizioni anomalie che il programma intende gestire, sia le routine all'interno del programma che devono assumere il controllo al verificarsi di quelle.

I parametri da inserire nel comando HANDLE per gestire gli inconvenienti piu' comuni nel BMS sono:

MAPFAIL (label)

Questa condizione si verifica, solo per operazioni di input, se i dati hanno lunghezza zero, oppure se il messaggio proviene da un 3270 non contiene la sequenza SBA-ADDR-ADDR. Usualmente e' il caso in cui l'operatore a terminale non immette dati e provoca l'invio della mappa mediante un ENTER o un tasto PF, oppure preme un tasto PA o il CLEAR.

L'azione implicita del CICS/VIS consiste nel chiudere in modo anomalo il task.

INVMPSZ (label)

La mappa indicata nel comando e' troppo grande rispetto alle dimensioni della pagina del terminale.

Vedremo in seguito che possono verificarsi condizioni di errore piu' complesse quando si utilizzano alcune funzioni particolari del BMS.

Possono verificarsi, poi, condizioni anomale diverse fra di loro ma in concomitanza. In tal caso solo la prima che il sistema riesce a identificare viene segnalata al programma, sempreche'.

naturalmente, sia stata opportunamente prevista nel comando HANDLE.

Nell'APRM, alla fine del capitolo 3.2-5 sul BMS e' possibile trovare l'elenco completo delle codizioni anomale e l'ordine con cui queste vengono testate.

5.6 GESTIONE DEI TASTI FUNZIONALI

In occasione di ogni operazione di input, il BMS imposta nel campo EIBAID del Execute Interface Block un AID (Attention Identifier). Esso indica quale metodo (praticamente: quale tasto) l'operatore ha usato per l'operazione di input, particolarmente in riferimento al terminale 3270. Esso puo' aver utilizzato il tasto ENTER, o la penna luminosa, o un tasto PF o PA. E' possibile controllare direttamente, dopo ogni RECEIVE MAP, il contenuto del campo EIBAID e, in funzione dell'identificatore, eseguire elaborazioni diverse all'interno del programma.

Un modo per esaminare il contenuto del campo EIBAID e' quello di fare uso di nomi standard forniti dal CICS/V5 e incorporabili nel programma con la COPY di DFHAID in WORKING-STORAGE SECTION.

Con questa COPY si ottiene la seguente lista di nomi:

| NOME SIMBOLICO | FUNZIONE PER |
|-------------------|------------------------------|
| | VIDEO 3270 |
| DFHCLEAR | TASTO CLEAR |
| DFHENTER | TASTO ENTER |
| DFHOPID | LETTORE DI TESSERA MAGNETICA |
| DFHTRIG | CAMPO TRIGGER |
| DEHPA1 | TASTO PA1 |
| DFHPA2 | TASTO PA2 |
| DFHPA3 | TASTO PA3 |
| DFHPEH | PENNA LUMINOSA |
| DFHPF1 | TASTO PF1 |
| DFHPF2 | TASTO PF2 |
| DFHPF24 | TASTO PF24 |

Per un elenco completo vedi APRM cap. 3.2-5

Per determinare il contenuto del byte AID, basta confrontare il campo EIBAID con il nome simbolico rappresentante il tasto da controllare. Per esempio per verificare il tasto ENTER, basta scrivere: IF EIBAID=DFHENTER THEN...

E' piu' comodo sfruttare la possibilita', offerta dal comando HANDLE AID, di codificare una routine specifica per ogni tasto da gestire.

Le routine ricevono il controllo solo dopo il completamento dell'operazione di input. La forma del comando e':

```
EXEC CICS HANDLE AID opzione(label) opzione(label)....
```

Possono essere specificate le opzioni seguenti:

1. qualunque tasto PF: PF1, PF2, FF3,, PF12
2. qualunque tasto PA: PA1, PA2, PA3
3. CLEAR o ENTER
4. LIGHTPEN
5. OPERID (lettere di tessera magnetica)
6. ANYKEY: qualunque PF; qualunque PA; CLEAR
7. TRIGGER

Un comando effettivo avra', ad esempio, la forma seguente:

```
EXEC CICS HANDLE AID PA1 (ROUTINE1)  
PA2 (ROUTINE2)
```

Nello stesso comando e' possibile specificare fino a 12 opzioni. Qualora l'operatore utilizzi un tasto che non compare nel comando, il programma riprendera' il controllo all'istruzione immediatamente successiva al comando di input.

L'efficacia del comando e' limitata al programma, e non si estende a programmi diversi all'interno dello stesso task.

Nel Cap. 3.2-2 dell'APRM puoi trovare informazioni dettagliate sull'uso del comando HANDLE AID.

Il comando HANDLE AID prevale su qualsiasi HANDLE CONDITION. Percio', se in un'operazione di input sono attivati entrambi i comandi, prendera' il controllo la routine di HANDLE AID.

PARTE SECONDA

5.7 FUNZIONI DEL TERMINAL CONTROL

Il TC eseguito come task all'interno del CICS/V5, viene definito task di sistema per distinguerlo dai task applicativi legati alle transazioni; e' il task a piu' alta priorita', e quindi sempre il primo ad avere il controllo dal Task Control Program (KCP).

Il TC ha due funzioni distinte:

1. Gestisce le linee e i terminali (della "rete"). In questa fase:

- Esegue il polling sui terminali, ricercando le transazioni ed i dati relativi da trasferire all'interno della regione CICS/V5 per l'elaborazione.
- Richiede al KC, quando necessario, la creazione di un task a fronte di ogni transazione.
- Dopo che il task e' stato creato, rende ad esso disponibili i dati della transazione.
- Se il task prevede l'invio di una risposta al terminale, esegue l'addressing ed invia i dati quando il terminale e' disponibile per la ricezione.

2. Fornisce i servizi che le applicazioni gli richiedono tramite i comandi CICS appositi.

Sottolineiamo la circostanza che durante l'elaborazione di un task, un solo terminale e' connesso al task; il TC e' in grado di sapere quali terminali sono collegati ai vari task, per cui il programmatore non dovrà preoccuparsi di individuare di volta in volta il terminale connesso; tale legame viene mantenuto automaticamente.

Tutti i dati destinati o provenienti da un terminale sono posti dal TCP in una TIOA (Terminal Input/Output Area) acquisita dinamicamente dal CICS.

5.8 COMANDI DI TC

I Comandi piu' importanti, e quindi di uso piu' frequente, con cui si richiedono i servizi di TC sono:

SEND per inviare dati ad un terminale

RECEIVE per ricevere dati da un terminale

WAIT TERMINAL per sincronizzare un task con una operazione di I/O su un terminale.

Vi sono altri comandi usati per scopi particolari, e che vanno quindi al di là degli obiettivi di questo corso; una loro illustrazione dettagliata si può trovare al cap. 3.3 dell'APRM.

In questo Capitolo 3.3 dell'APRM è possibile trovare tutti i comandi e tutte le opzioni per tutti i terminali supportati dal CICS/VS.

Percio' e' bene conoscere a fondo le caratteristiche dei terminali della propria rete, per poter scegliere opportunamente tra le molte opzioni disponibili.

5.8.1 Invio dati ad un terminale

Il formato generale del comando è:

EXEC CICS SEND FROM(nome di campo) LENGTH(numero)
(WAIT)

FROM (nome di campo) identifica l'area di memoria all'interno del programma in cui è stato costruito il messaggio.

LENGTH (numero) fornisce la lunghezza del messaggio.

Quando l'operazione di output è iniziata, il controllo ritorna al programma applicativo, che potrà proseguire l'elaborazione, ma che non potrà eseguire né una SEND né una RECEIVE prima che l'operazione di output iniziale sia completata.

E' possibile, però, fermare l'esecuzione del programma, fino al completamento del comando SEND, utilizzando l'opzione WAIT, che appunto condiziona la riassunzione del controllo da parte del programma al momento in cui l'operazione di output è terminata.

La stessa funzione dell'opzione WAIT può svolgerla il comando:

EXEC CICS WAIT TERMINAL

5.8.2 Ricezione dati da un terminale

Quando un programma applicativo intende avere a propria disposizione i dati inviati da un terminale (ricordiamo che è il TC a riceverli), deve fornire il comando RECEIVE:

EXEC CICS RECEIVE (INTO(nome-campo) LENGTH(nome-campo)
/SET (pointer))

Il comando potra' comprendere alcuni altri parametri dipendenti dal tipo di terminale.

Sappiamo che il messaggio viene ricevuto dal TCP in una TIOA. Il comando RECEIVE rende il messaggio disponibile al task, o nella TIOA direttamente (opzione SET), o trasferendolo in una area di lavoro del programma (opzioni INTO e LENGTH).

Il comando perciò puo' lavorare in due modi distinti:

Tecnica MOVE: i dati vengono spostati dalla TIOA in un'area definita in WORKING-STORAGE SECTION.

Tecnica LOCATE: i dati sono disponibili nella TIOA in cui il TCP li ha ricevuti.

La tecnica LOCATE e' un poco piu' complessa di quella MOVE, e richiede al programmatore la capacita' di gestire indirizzi di aree di memoria esterne al programma (vedi il Cap.13). In questo corso tratteremo preferenzialmente la Tecnica Move.

La Tecnica Move viene utilizzata specificando l'opzione INTO(nome di campo) nel comando, dove il "nome di campo" e' quello di un'area definita in WORKING-STORAGE SECTION.

Nell'opzione LENGTH deve essere specificato un campo di due byte binari contenente la lunghezza massima del messaggio che ci si aspetta di ricevere, per evitare di ricoprire aree di memoria contigue a quella in cui viene ricevuto. Se la lunghezza effettiva del messaggio e' superiore alla massima specificata, il messaggio viene troncato e si verifica la condizione anomala di LENGTHERR. Quando il messaggio sara' stato ricevuto, nel campo specificato nella opzione LENGTH comparira' la sua lunghezza effettiva, che potra' essere, eventualmente, quella originaria, prima di un troncamento.

Note: Nell'esecuzione di un comando RECEIVE viene sempre assicurato il completamento dell'operazione di input, prima che il programma possa proseguire l'elaborazione (WAIT implicito).

Nel già citato Cap. 3.3 dell'APRM sono elencate anche tutte le condizioni anomale che si possono verificare, in seguito ad un comando di TC, per ogni tipo di terminale.

5.9 NOTE SU SEND E RECEIVE

Un programma contenente delle istruzioni SEND/RECEIVE o SEND MAP/RECEIVE MAP, quando viene eseguito si fermerà sempre ad ogni istruzione RECEIVE o RECEIVE MAP, finché l'operatore non preme un tasto di invio del terminale.

Ricordiamo che i tasti di invio sono:

- **ENTER**
- **CLEAR**
- da PF1 a PF24
- PA1, PA2 e PA3

Questo permette all'operatore di utilizzare, senza limiti di tempo, la mappa eventualmente presente al terminale.

D'altra parte c'e' da notare che due comandi di SEND, o di SEND MAP, ravvicinati saranno eseguiti l'uno dopo l'altro ed impediranno all'operatore di vedere l'esito del primo.

Un modo per ovviare a questo inconveniente e' di codificare, tra i due comandi SEND/SEND MAP, il comando

EXEC CICS RECEIVE END-EXEC

senza opzioni, che ha l'effetto di interrompere l'esecuzione del programma dopo la prima istruzione di SEND/SEND MAP.

5.10 SOMMARIO

In questo capitolo abbiamo visto i comandi necessari per inviare e ricevere dati con l'uso di:

1. una singola mappa (comandi BMS)
2. comandi di Terminal Control

Quando inviamo un comando di "input mapping" il TCP legge i dati del terminale e li muove nella TIDA. Il BMS rimuove i caratteri di controllo e mette i dati formattati in un buffer.

I dati possono essere elaborati nel buffer o nell'area di lavoro del programma.

In questo capitolo abbiamo discusso solo l'elaborazione nella Working-Sorage Section di un programma COBOL. Infatti abbiamo visto che l'accesso ai dati e' garantito dai nomi suffissati della mappa simbolica (copiata nel programma da un'istruzione COPY).

Il comando BMS per l'"input mapping" e' RECEIVE MAP.

Questo comando, utilizzando i formati della mappa fisica e di quella simbolica, mette tutti i dati modificati nell'area di lavoro definita dal programma applicativo.

Quando, invece, il programma vuole inviare dati ad un terminale, deve far uso del comando di "output mapping" del BMS: SEND MAP.

Eseguendo questo comando il BMS mette gli appropriati caratteri di controllo del terminale all'inizio di ogni campo dati. La stringa dati risultante viene quindi messa nella TIOA, che, infine, e' trasmessa al terminale dal TCP.

Ad ogni campo definito in una mappa e' associato un byte per l'attributo. Questo puo' essere temporaneamente modificato dal programma applicativo, prima di inviare la mappa a terminale, facendo uso della lista dei nomi simbolici per gli attributi (codificata nel programma dell'istruzione di COPY di DFHBMSCA).

Durante l'esecuzione di comandi SEND MAP e RECEIVE MAP possono verificarsi diverse condizioni eccezionali (o anomale). Tra queste abbiamo visto la condizione di MAPFAIL (mappa senza dati) e quella di INVMPSZ (dimensione invalida di mappa).

Il byte AID permette di identificare il metodo scelto dall'operatore per iniziare il trasferimento dei dati dal terminale al CICS/VS.

Questo byte puo' essere testato esaminando il campo EIBAID del blocco EIB, per un certo valore, oppure codificando il comando HANDLE AID.

Negli ultimi paragrafi abbiamo visto anche i comandi del Terminal Control Program (SEND e RECEIVE), che permettono al programma applicativo di comunicare direttamente con i terminali, senza far uso delle mappe BMS.

Per ulteriori informazioni riguardo i vari comandi elencati siamo a vostra disposizione presso il servizio Client Support di IBM Italia.

IBM Italia ha sempre al vostro servizio:

Numero Verde 177 70000000

Ogni giorno dal lunedì al venerdì dalle 08:00 alle 18:00, esclusi i giorni festivi, le giornate di sciopero e le giornate di rientro da ferie. I numeri 177 70000000 e 177 70000001 sono attivati solo per chi ha già effettuato la registrazione sul sito www.italia.ibm.com.

Per ulteriori informazioni sui servizi IBM Italia, visitate il sito www.italia.ibm.com.

Per ulteriori informazioni sui servizi IBM Italia, visitate il sito www.italia.ibm.com. Per ulteriori informazioni sui servizi IBM Italia, visitate il sito www.italia.ibm.com.

Per ulteriori informazioni sui servizi IBM Italia, visitate il sito www.italia.ibm.com.

Per ulteriori informazioni sui servizi IBM Italia, visitate il sito www.italia.ibm.com.

Per ulteriori informazioni sui servizi IBM Italia, visitate il sito www.italia.ibm.com.

6.1 INTRODUZIONE

In questo capitolo vedremo le funzioni del Program Control Program (PCP), che e' il modulo del CICS responsabile di gestire i programmi. In particolare di caricarli in memoria, se necessario, per l'esecuzione, trasferire loro il controllo e restituirlo al CICS/V5 alla fine della loro esecuzione.

Analizzeremo le due funzioni del PCP (LINK e XCTL) che permettono di trasferire il controllo da un programma ad un altro, con il passaggio eventuale di dati.

Infine faremo alcune considerazioni sulle tecniche di programmazione conversazionale e pseudo-conversazionale.

6.2 CONSIDERAZIONI APPLICATIVE

E' ormai consuetudine, quando si scrivono programmi per grosse applicazioni, dividere logicamente il programma in sezioni e, per ognuna di queste, codificare un modulo separato.

In un programma "batch" tali moduli sono poi assiemati dal Linkage Editor in un modulo unico eseguibile che viene catalogato come unico membro di una libreria di programmi.

Questo metodo sarebbe troppo dispendioso in ambiente "on-line", perchc richiederebbe la disponibilita' immediata di una quantita' di memoria eccessiva rispetto alle effettive necessita' del programma (in fase di esecuzione).

Per ovviare a questo, in ambiente CICS/V5 tutti i singoli moduli sono catalogati in libreria come programmi indipendenti e solo in fase di esecuzione vengono opportunamente correlati.

In pratica e' il PCP che, a seconda delle richieste, pone il controllo da un programma ad un altro e tiene aggiornata la situazione dei programmi, modificando le relative entrate nella PPT (si ricordi il "contatore d'uso" dei programmi).

Sono tre i comandi CICS/V5 che possono essere utilizzati per chiedere al PCP di passare il controllo da un programma ad un altro:

1. il comando LINK;
2. il comando XCTL;

3. il comando RETURN.

Vi sono poi due comandi di PCP che consentono di gestire le tabelline:

1. il comando LOAD;
2. il comando RELEASE.

6.3 IL COMANDO LINK

In un programma applicativo il comando LINK viene utilizzato per chiamare in esecuzione un altro programma e, alla fine di questo, riprendere il controllo.

Il formato più semplice del comando è:

```
EXEC CICS LINK PROGRAM (nome)
```

dove "nome" è il nome, descritto in PPT, del programma chiamato che deve prendere il controllo e va scritto tra apici: "nome".

In particolare, il programma chiamante (cioè quello che invia il comando LINK) si ferma, ma non termina. Il programma chiamato viene caricato in memoria (solo se è necessario), quindi riceve il controllo ed esegue le sue istruzioni finché non incontra un comando RETURN (vedi par. 6.5). A quel punto, il controllo torna al programma chiamante che riprende l'esecuzione all'istruzione successiva al comando LINK.

Per definizione, il programma chiamato con un comando LINK è ad un livello logico più basso del suo chiamante.

Per il concetto di "livello logico" e per le relative implicazioni, si guardi l'inizio del capitolo 4.4 dell'APRM.

6.4 IL COMANDO XCTL

Questo comando è usato per trasferire definitivamente il controllo da un programma applicativo ad un altro.

Il formato più semplice del comando è:

```
EXEC CICS XCTL PROGRAM (nome)
```

dove "nome" è il nome, definito in PPT, del programma che deve prendere il controllo e va scritto tra apici.

Il programma chiamante viene rilasciato dal task e il programma chiamato, appena caricato in memoria (se non già presente), viene eseguito. Quando incontra un comando RETURN (vedi par. 6.5), non restituisce il controllo al programma chiamante, ma al pro-

gramma a livello logico immediatamente superiore o al CICS/VIS se non sono stati inviati precedenti comandi di LINK.

Da questo si intuisce che la XCTL e' l'ultima istruzione eseguibile dal programma chiamante.

Per definizione il programma chiamato con un comando XCTL e' allo stesso livello logico del suo chiamante.

6.5 IL COMANDO RETURN

Il comando RETURN e' un altro comando gestito dal PCP e serve per restituire il controllo dal programma in esecuzione al programma di livello logico immediatamente superiore.

Esso rappresenta il modo normale per chiudere un programma e, perciò, deve essere sempre codificato. Con questo comando si possono presentare due casi:

1. Se il programma in esecuzione e' al primo livello logico, il comando RETURN, oltre che chiudere il programma, restituisce il controllo al CICS/VIS e chiude anche il task.
2. Se il programma in esecuzione e' ad un livello logico più basso del primo, il comando RETURN chiude il programma (ma non il task) e restituisce il controllo al programma che aveva inviato il comando di LINK e che risiede il livello logico sopra.

Il formato più semplice del comando e':

```
EXEC CICS RETURN (TRANSID (codice))
```

dove e' presente l'opzione facoltativa TRANSID. Questa permette, all'atto della chiusura del task, di specificare un codice di transazione di un nuovo task da associare a quel terminale. Questo metodo (detto "Concatenamento di Transazioni"), che evita all'operatore di specificare i quattro caratteri identificatori della transazione, e' utile quando si debba utilizzare lo stesso terminale più volta per la stessa applicazione. Esempio

```
EXEC CICS RETURN TRANSID ('TRXX')  
END-EXEC
```

dove TRXX e' il codice di transazione definito in PCT.

NOTA: In ambiente CICS/VIS un programma COBOL non deve assolutamente chiudere con l'esecuzione dell'istruzione STOP RUN, perché questa provocherebbe un ritorno al Sistema Operativo e non al CICS/VIS.

Poiché il compilatore COBOL può generare automaticamente una istruzione di STOP RUN, il programmatore

deve sempre codificare un comando EXEC CICS RETURN o una istruzione GOBACK alla fine del programma.

6.6 CARICAMENTO DI UNA TABELLA

Per caricare una tabella di valori, necessari all'esecuzione di un programma, possiamo usare il comando LOAD:

```
EXEC CICS LOAD PROGRAM (nome)
      SET (pointer)
      (LENGTH (nome-area)/
      FLENGTH (nome-area))
      (HOLD)
```

PROGRAM Il nome indica la tabella da caricare e deve avere un'entrata in PPT.

SET Il CICS/VS memorizza l'indirizzo d'inizio della tabella caricata nell'argomento "pointer" dell'opzione SET. Il campo "pointer" deve essere una voce binaria definita come un puntatore (vedi il Cap.13).

LENGTH Indica il nome di area dove il CICS/VS scriverà la lunghezza della tabella caricata. Il campo deve essere una mezza voce binaria.

FLENGTH In alternativa a LENGTH. Ha lo stesso significato, ma il campo deve essere una voce binaria.

HOLD Si riferisce all'area di memoria occupata dalla tabella caricata.

- Se è omesso, la memoria viene rilasciata automaticamente alla fine del task che ha emesso il comando LOAD.

- Se è incluso, la tabella rimane in memoria finché non è emesso il comando:

```
EXEC CICS RELEASE PROGRAM (nome)
```

L'opzione HOLD è utile per mantenere una tabella in memoria per l'intera durata del CICS/VS.

6.7 LA CONDIZIONE ANOMALA PGMDERR

Come abbiamo già detto, quando un programma applicativo invia un comando:

```
LINK PROGRAM (nome)
XCTL PROGRAM (nome)
LOAD PROGRAM (nome)
RELEASE PROGRAM (nome)
```

il nome del programma indicato deve essere definito in PPT. Se questo non e' vero, si verifica la condizione di "errore di identificazione del programma" (PGMIDERR), che manderà in abend la task. Per evitare l'abend bisogna codificare l'opportuna HANDLE CONDITION prima di eseguire un comando LINK, XCTL, LOAD o RELEASE.

6.8 PASSAGGIO DI DATI AD ALTRI PROGRAMMI

Spesso, per esigenze applicative, c'e' necessita' di trasferire dati da un programma ad un altro.

In ambiente CICS sono disponibili diverse funzioni che permettono la gestione dello scambio di dati sia tra programmi che fra transazioni concatenate.

In questo paragrafo vedremo le possibilite' offerte dai comandi di PC LINK, XCTL e RETURN.

Il formato completo dei comandi LINK e XCTL e' il seguente:

```
EXEC CICS LINK PROGRAM (nome)
    (COMMAREA (area-dati)
     LENGTH (numero))
```

```
EXEC CICS XCTL PROGRAM (nome)
    (COMMAREA (area-dati)
     LENGTH (numero))
```

Quando un programma deve trasferire il controllo ad un altro programma e, contemporaneamente, passargli dei dati, il programma chiamante deve indicare (nel comando LINK o XCTL) un'"area di comunicazione" apposita (opzione COMMAREA).

L'argomento "area-dati" dell'opzione COMMAREA e' il nome dell'area di lavoro in cui sono stati messi i dati da trasferire (in WORKING-STORAGE per il COBOL).

L'opzione LENGTH, obbligatoria per il COBOL, rappresenta la lunghezza dei dati da passare.

Il programma chiamato deve, a sua volta, essere predisposto per ricevere i dati dell'"area di comunicazione". Essendo questa un'area esterna al programma chiamato, dobbiamo definire:

COBOL: il primo campo in LINKAGE SECTION a livello 01 con nome riservato DFHCOMMAREA.

L'area di ricezione definita nel programma chiamato non necessita che sia della stessa lunghezza dell'area di comunicazione definita nel chiamante.

Per esempio, nel caso che si voglia disporre solo della prima parte dei dati trasferiti, l'area di ricezione puo' essere piu' piccola.

Comunque, questa non deve mai essere piu' lunga dell'area di comunicazione, onde evitare risultati imprevedibili.

In particolare, il passaggio dati con l'uso dell'opzione COMMAREA, in un comando LINK, permette al programma chiamato, oltre che ricevere i dati dell'area di comunicazione, anche di modificarli e di renderli cosi' disponibili al programma chiamante, quando questo riprendera' il controllo.

Poiche', pero', con un comando LINK, la COMMAREA non viene fisicamente passata al programma chiamato (ma viene fornito solo l'indirizzo), il programma chiamante dovrà, se necessario, creare un'apposita struttura in COMMAREA per gli eventuali dati di ritorno creati dal programma chiamato.

Abbiamo detto che una COMMAREA e' un'area interna al programma chiamante: questo non e' sempre vero!

Infatti se un programma A chiama un programma B con un comando LINK e gli passa la COMMAREA AAAA, e se il programma B, a sua volta, chiama il programma C e gli passa la stessa area AAAA, allora la COMMAREA AAAA e' interna al programma A, ma esterna a B oltre che a C.

Per sapere la dimensione dell'area di comunicazione, il programma chiamato deve riferirsi al contenuto di un campo del blocco EIB chiamato EIBCALEN.

E' proprio tale campo EIBCALEN che fornisce l'ampiezza dell'area di comunicazione del programma chiamante, in particolare:

- EIBCALEN assume valore zero quando nessun dato viene passato.
- Se il programma chiamante ha trasferito dati, EIBCALEN conterrà il valore presente nell'opzione LENGTH del comando di PC relativo.

Vediamo l'esempio seguente:

```
=====
E S E M P I O   C O B O L
=====

      .. PRogramma CHIAMANTE      ..
```

IDENTIFICATION DIVISION.
PROGRAM-ID. PROGR1.

WORKING-STORAGE SECTION.
01 TRANS-INFO.
 02 FLD1 PIC X(4).

```
PROCEDURE DIVISION.  
    MOVE 'INV'C TO FLD1.  
    EXEC CICS LINK PROGRAM('PROG2')  
        COMMAREA(TRANS-INFO)  
        LENGTH(4)  
    END-EXEC.
```

PROGRAMMA CHIAMATO

```
IDENTIFICATION DIVISION.  
PROGRAM-ID. PROG2.
```

```
LINKAGE SECTION.  
01 DFHCOMMAREA.  
02 FUNCTION PIC X(4).
```

```
PROCEDURE DIVISION.  
    IF EIBCALEN > 0 THEN IF FUNCTION='INV'C' .....
```

Esaminiamo insieme questo esempio:

1. Il programma chiamante (PROG1) ha definito un'area di comunicazione che contiene un campo di quattro caratteri, chiamato FLD1.
2. Un certo dato è stato mosso in FLD1 (INV'C).
3. Un comando LINK è stato dato dal programma chiamante fornendo il nome dell'area di comunicazione e, per COBOL la sua ampiezza.
4. Il programma chiamato (PROG2) contiene la definizione di un'area di lavoro (FUNCTION), ad esso accessibile e contenente i dati passati.
5. Viene esaminato il contenuto del campo EIBCALEN: se il suo valore è diverso da zero si procede nell'elaborazione dei dati.

Si noti che una qualunque modifica ai dati provocata dal programma chiamato si ripercuote in analoga modifica agli stessi dati all'interno del programma chiamante, una volta che questi riacquisti il controllo.

Per l'analisi delle possibili variazioni, il programma chiamante dovrà semplicemente far riferimento all'area di comunicazione il cui nome compare come argomento dell'opzione COMMAREA.

Notizie più dettagliate sull'argomento si possono trovare nel capitolo 4.4 dell'APRM.

Avrete certo notato, nell'esempio, che non è stato incluso, nel programma chiamato, il comando RETURN. Questo comando, certamente necessario quando si sia fatto uso di una LINK, richiede qualche altra considerazione.

Abbiamo visto che il comando RETURN cede il controllo ad un programma di livello logico immediatamente superiore (al CICS/VIS, in particolare, se il programma in questione è quello a livello logico più alto); possiamo però includere in questo comando le opzioni LENGTH e COMMAREA alla stessa stregua di quanto visto per i comandi LINK e XCTL, cioè:

```
EXEC CICS RETURN (TRANSID (codice)
                   (COMMAREA (nome dato) LENGTH (numero)))
```

ma ricordiamoci che le opzioni COMMAREA e LENGTH possono essere adottate solo quando il comando RETURN restituisce il controllo al CICS/VIS e quando comprende, specificata, la opzione TRANSID. In questo modo si passano dati ad un nuovo task che sta per partire; agendo in modo differente si provoca la condizione anomala di richiesta invalida; questa è gestibile coll'opzione INVREQ ma, se lasciata a carico del CICS/VIS, causa la chiusura anomala del task.

6.9 FINE ANORMALE DI UN TASK

Un task può chiudere essenzialmente in due modi:

1. CHIUSURA NORMALE: tutto è andato bene e il controllo torna al CICS/VIS.
2. CHIUSURA ANORMALE: si è verificato un errore durante l'elaborazione di un task e si rende necessaria qualche tipo di azione; vi sono due possibilità:
 - Lasciare che il CICS/VIS chiuda il task (ABEND di transazione)
 - Intraprendere delle azioni correttive diverse dalla chiusura anomala del task. Questa possibilità si realizza codificando un opportuno comando di PC che attivi, a livello ABEND di programma, una 'exit' ad una routine scritta dall'utente.

Il comando è:

```
EXEC CICS HANDLE ABEND
```

(PROGRAM (nome)/LABEL (nome)/CANCEL/RESET)

Questo comando viene usato per attivare, cancellare o riattivare una 'exit'.

NOTE:

- L'opzione LABEL e' usata solo col COBOL
- Per cancellare o riattivare una 'exit' si rimpiazzi la parola-chiave PROGRAM o LABEL con CANCEL o RESET. Questa azione viene intrapresa nel caso si sia precedentemente codificato nel programma un comando di HANDLE ABEND per l'attivazione di una 'exit'.
- Ci puo' essere solo una 'exit' attiva per ogni livello logico. Questo significa che ogni nuovo HANDLE ABEND disattiva ogni altro comando HANDLE ABEND definito precedentemente in un programma di uguale livello logico.
- Quando un task chiude in modo anomale il CICS/VIS verifica la presenza di 'exit' dapprima allo stesso livello logico e successivamente, se del caso, ai livelli logici piu' alti; in ogni caso la prima exit routine attivata che si incontra assume il controllo.

Per riassumere e rivedere quanto detto finora, rifacciamoci alla figura 6.1.

In fase di prova dei programmi puo' essere utile interrompere volontariamente l'elaborazione di un task (in modo anomale) e stampare la memoria occupata dalle aree legate alla transazione (dump). Questo e' possibile codificando il seguente comando:

EXEC CICS ABEND (ABCDE (nome)) (CANCEL)

L'argomento associato alla opzione ABEND (non piu' di quattro caratteri fra apici) e' il codice che verrà usato per identificare il 'dump' associato alla transazione. CANCEL, in questo comando, disattiva tutte le 'exit' a qualunque livello nel task.

L'unica condizione anomala che si puo' verificare nella esecuzione del comando di HANDLE ABEND e' gestibile dall'opzione PGMDERR; essa si verifica quando il programma, il cui identificatore compare come argomento della opzione PROGRAM, non si trova nella PPT o risulta disabilitato.

Nella figura 6.1 sono rappresentati graficamente i possibili comportamenti del task in caso di abend.

6.10 TRANSAZIONI CONVERSATORIALI E PSEUDO

Nel Cap. 1 abbiamo definito il "Task Utente" come l'unità di lavoro dell'utente.

In ambiente CICS/VS si parla invece di "transazioni" e di "task".

Per "transazione" s'intende uno o più programmi richiamabili in esecuzione da un codice di identificazione (TRANSID), disponibile a più utenti.

Per "task" s'intende l'unità di lavoro del CICS/VS, cioè l'esecuzione effettiva di una transazione richiesta da un particolare utente.

Il dialogo operatore-elaboratore, descritto in un "task utente", può essere realizzato, in ambiente CICS con una o più transazioni e, in base alla scelta fatta dal programma applicativo, usando una tecnica conversazionale oppure una tecnica pseudoconversazionale.

Come vedremo, queste due tecniche differiscono nettamente nell'uso delle risorse CICS/VS, ma permettono di presentare il dialogo in modo del tutto equivalente per l'operatore.

6.10.1 Transazione Conversazionale

Vediamo come si presenta il flusso di una transazione conversazionale:

1. Come tutte le transazioni, una transazione conversazionale può essere iniziata da un operatore digitando a terminale un codice di transazione o un tasto funzionale (PA o PF) opportuno.
2. A fronte di questo il CICS/VS crea un task e carica il programma indicato nella PCT.
3. Il programma elabora gli eventuali dati comunicati dall'operatore e, poi, invia nuovi dati all'operatore, utilizzando di solito una mappa. Quindi si metta in attesa di una risposta.
4. L'operatore legge la mappa, pensa cosa deve fare e, infine, introduce dei dati.
5. Il programma riceve i dati e riprende l'elaborazione. Quindi spedisce all'operatore nuovi dati e si mette in attesa di una risposta.
6. Si ripetono i passi 4. e 5. fino a quando il task chiude, restituendo il controllo al CICS/VS.

Perciò, in un flusso conversazionale, c'è un solo task in esecuzione che viene sospeso per tutto il tempo necessario all'operatore per pensare e digitare una risposta (vedi fig. 6.2).

Cosa succede se questi, per un motivo qualsiasi non e' in grado di proseguire il colloquio?

Il task rimane attivo, quanto a dire che tutte le risorse CICS/VS associate ad esso (ad esempio aree di lavoro; records letti per aggiornamento; area di memoria occupata dal programma) non sono disponibili per altri task, fintanto che l'operatore non completa la transazione in sospeso, oppure fino a quando non sono rilasciati esplicitamente con apposito comando.

Un modo per rimediare a questo possibile inconveniente e' quello di codificare in PCT un parametro di "time-out", che indichi l'intervallo di tempo massimo per formulare una risposta, trascorso il quale il CICS/VS provochera' la chiusura anomala del task.

I vantaggi che offre una tecnica conversazionale sono:

- **possibilita' di controllare sempre l'attivita' dell'operatore al terminale;**
- **massima protezione nei confronti di qualunque tipo di errore durante operazioni di variazioni sui dati.**

Vedi la figura 6.2.

6.10.2 Transazione Pseudoconversazionale

Vediamo ora il flusso di una transazione pseudoconversazionale, che, come abbiamo detto, dal punto di vista dell'operatore, non presenta nessuna differenza rispetto ad una transazione conversazionale.

1. Per iniziare una transazione pseudoconversazionale vale quanto detto prima: un operatore digita un codice di transazione o un tasto funzionale.
2. Il CICS/VS crea un task e carica il programma indicato nella PCT.
3. Il programma elabora gli eventuali dati comunicati dall'operatore, invia una mappa al terminale e, subito dopo, esegue un comando RETURN con l'opzione TRANSID. In questo modo provoca la chiusura del task ed imposta un (nuovo) codice di transazione in grado di iniziare un nuovo task collegato allo stesso terminale.
4. L'operatore legge la mappa, pensa cosa deve fare e, infine, introduce dei dati. Solo dopo l'invio dei dati, parte il nuovo task.
5. Si ripetono i passi 2., 3. e 4. fino a quando l'ultimo task chiude senza usare l'opzione TRANSID nel comando RETURN e restituisce definitivamente il controllo al CICS/VS.

Percio', in un flusso pseudoconversazionale, si provoca la creazione di tanti task quante sono le coppie di messaggi inviati all'elaboratore e di risposte spedite all'operatore, che esauriscono la transazione (e il dialogo), impedendo cosi' che i task restino in attesa sui tempi dell'operatore.

Quindi questa tecnica ha il grosso vantaggio di non tenere impegnate le risorse CICS/V5 associabili ai task, durante i periodi necessari all'operatore per riflettere e digitare.

In pratica ogni task si apre con un comando RECEIVE e si chiude subito dopo il primo comando SEND, come esemplificato in figura 6.3.

Gli svantaggi che offre una tecnica pseudoconversazionale sono:

- Laboriosa gestione degli errori operativi e necessita' di ricorrere a supporti di memorizzazione intermedi come il Transient Data o il Temporary Storage (vedi i Capitoli 9 e 10).
- Limitata protezione dei dati nei confronti di abend, cadute di sistemi e simili, dato che il supporto offerto dalle funzioni CICS/V5 non si estende oltre la vita di un task, e quindi la protezione dei dati non viene fatta a livello di "transazione".

In genere la codifica di un programma con tecnica pseudoconversazionale necessita di maggiore impegno da parte del programmatore, soprattutto se e' necessario realizzare un meccanismo per garantire l'integrita' dei dati.

Durante lo svolgimento di una transazione con tecnica pseudoconversazionale, si presentera' spesso la necessita' di trasferire dati da un task al successivo, in modo che l'elaborazione possa regolarmente proseguire.

Per questo ricordiamo (rivedi la fine del paragrafo 6.8) che il comando RETURN, oltre all'opzione TRANSID, permette l'uso dell'opzione COMMAREA, che consente, attraverso l'uso di un'area di comunicazione, di passare dati al task che sta per partire.

Vedi la figura 6.3.

6.11 SOMMARIO

Attraverso l'uso di una serie di comandi, il PCP e' in grado di eseguire le seguenti funzioni:

- Caricare un programma a livello logico piu' basso e dargli il controllo (comando LINK). Il controllo tornera' in seguito al programma chiamante.

- Caricare un programma allo stesso livello logico e dargli il controllo (comando XCTL). Il controllo non tornerà più al programma chiamante.
- Chiudere un programma in modo normale (comando RETURN con o senza opzione TRANSID) oppure in modo forzato (comando ABEND con o senza opzione ABCODE).
- Caricare una tabella o un programma (comando LOAD).
- Passare dati tra "programmi applicativi o fra task diversi (opzione COMMAREA nei comandi LINK, XCTL e RETURN TRANSID).

Abbiamo infine visto come una transazione può essere realizzata con tecnica conversazionale o pseudoconversazionale, senza peraltro sembrare diversa all'operatore.

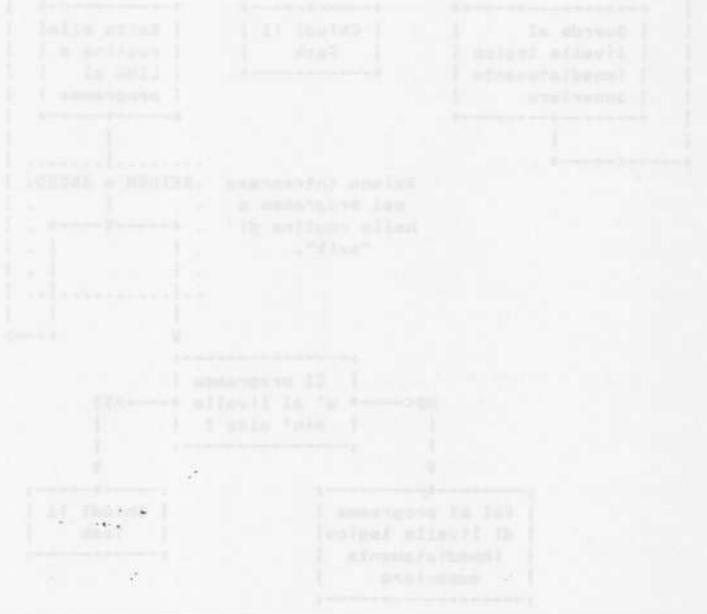


Diagramma delle transazioni di conversazione PDS - E di conversazione

6.12 FIGURE

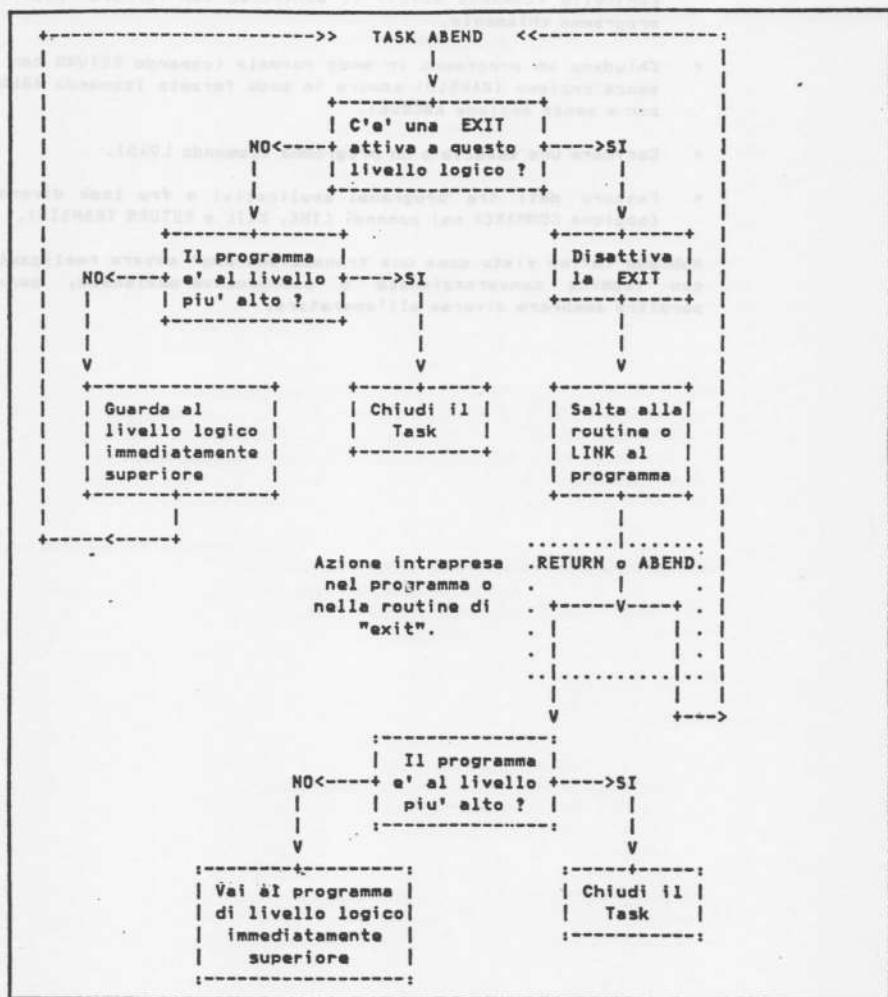


Figura 6.1 - Chiusura di un Task in caso di Abend.

```

+-----> Inizio TASK
      RECEIVE
      Elabora
      .
|Data |----> .
|Set |<---- .
+-----+
      SEND
      RECEIVE
      Elabora
T R A H S A Z I O N E
      .
|Data |----> .
|Set |<---- .
+-----+
      SEND
      RECEIVE
      Elabora
      .
|Data |----> .
|Set |<---- .
+-----+
      SEND
      RETURN
+-----> Fine TASK

```

Figura 6.2 - Ambiente Conversazionale

```

+-----> Inizio TRANSAZIONE
| +-----> Inizio TASK A
| |
| | RECEIVE
| | Elabora
| +----+
| | |Data |----> .
| | |Set |<---- .
| | +----+
| | SEND
| | RETURN TRANSID B
T +-----> Fine TASK A
R +-----> Inizio TASK B
A | |
| RECEIVE
| Elabora
S | +----+
| | |Data |----> .
| | |Set |<---- .
| | +----+
| | SEND
| | RETURN TRANSID C
Z +-----> Fine TASK B
+-----> Inizio TASK C
I | |
| RECEIVE
| Elabora
N | +----+
| | |Data |----> .
| | |Set |<---- .
| | +----+
| | SEND
| | RETURN
E +-----> Fine TASK C
+-----> Fine TRANSAZIONE

```

Figura 6.3 - Ambiente Pseudo-Conversazionale

7.1 INTRODUZIONE

In questo capitolo vedremo come e' possibile programmare in ambiente on-line, utilizzando le funzioni offerte dal DL/I per accedere ai Data Bases.

In particolare studieremo quali sono i "comandi DL/I" che e' possibile utilizzare in ambiente CICS/V5, insieme con il linguaggio COBOL, e come il traduttore di comandi EXEC del CICS/V5 svolgera' analoghe funzioni anche nei confronti dei comandi DL/I.

7.2 DEFINIZIONI E VISTA DI UN DATA BASE

Sappiamo che la struttura di un Data Base e le informazioni relative ad ogni suo segmento (cioe' il dispositivo fisico di memorizzazione, il metodo d'accesso usato dal DL/I e le relazioni fra i segmenti) sono descritte, indipendentemente da ogni programma applicativo e dal contenuto fisico dei record di Data Base, in un blocco DBD (Data Base Description), opportunamente generato per ogni Data Base e memorizzato in apposita libreria.

In fase di elaborazione di un Data Base, puo' essere necessario, per comodita' applicativa o per ragioni di sicurezza, limitare l'accesso solo a certi tipi di segmenti del Data Base.

Cioe' si tratta di assegnare al programma applicativo una "vista del Data Base", che rappresenti quella porzione dell'intera struttura gerarchica (segmenti e campi) cui esso e' sensitivo.

Ogni vista di Data Base viene rappresentata da un blocco PCB (Program Communication Block) al quale ogni programma in esecuzione deve far riferimento. E' ovvio che:

1. Un Data Base puo' originare piu' PCB.
2. Uno stesso PCB puo' essere utilizzato da programmi diversi.

Tutti i blocchi PCB che servono ad uno stesso programma applicativo vengono raggruppati insieme in un nuovo blocco PSB (Program Specification Block), che deve essere catalogato nell'apposita libreria.

In fase di esecuzione, ogni programma che usa il DL/I, prima di poter richiedere un qualsiasi servizio, deve indicare quale PSB intende utilizzare. Percio' per ogni programma DL/I deve esserci almeno un PSB generato e catalogato.

7.3 L'AMBIENTE DL/I

Il DL/I e' un prodotto programma che lavora come intermediario tra il programma applicativo ed uno (o piu') dei data base memorizzati su dispositivi esterni.

Prima che un programma contenente richieste al DL/I possa essere eseguito, l'Amministratore dei Data Base (DBA) deve predisporre opportunamente l'ambiente; in particolare deve:

- Generare i blocchi DBD per i data base da utilizzare;
- Generare i blocchi PSB per il programma applicativo;
- Preparare i files VSAM;
- Caricare i data base.

Solo allora il programma applicativo COBOL puo' essere tradotto, compilato, link-editato, catalogato in libreria ed, infine, eseguito.

7.4 IL PROGRAMMA CICS-DL/I

Prima di intraprendere la codifica di un programma DL/I online, il programmatore deve conoscere quali sono le informazioni da utilizzare nell'applicazione; in particolare deve sapere:

- Quali sono le funzioni e i dati che il programma utilizzerà.
- Quali sono i segmenti ed i campi interessati all'elaborazione.
- Quali sono i PSB da schedulare e quanti PCB contengono.
- Quali sono, per ogni data base, le caratteristiche e le restrizioni del metodo d'accesso.

Tutte le informazioni necessarie sono reperibili nella documentazione dell'applicazione (vedi cap. 1), ma sara' comunque opportuno contattare il DBA per maggiori dettagli sull'ambiente DL/I da lui predisposto e per l'eventualita' di utilizzare funzioni particolari del DL/I, come il "posizionamento multiplo".

7.5 I SERVIZI DL/I IN AMBIENTE ON-LINE

I servizi DL/I in ambiente on-line sono gli stessi disponibili in ambiente batch con, in piu', due funzioni tipiche:

1. la schedulazione del PSB (SCHEDULING)
2. il rilascio del PSB (TERMINATING)

7.5.1 SCHEDULING

le possibilta', da parte di un programma CICS/V5, di accedere ad un data base DL/I sono definite in un PSB creato appositamente dal Data Base Administrator DBA.

Il PSB contiene uno o piu' PCB che descrivono le possibilta' di accesso ai dati per ogni data base DL/I che debba essere acceduto dal programma applicativo.

Poiche' in ambiente on-line, i data base sono normalmente condivisi con altre applicazioni, sara' necessario controllare gli accessi simultanei ai dati.

Per questo un programma CICS/V5, prima di accedere a un data base, deve dichiarare al DL/I l'intenzione di usarlo, cioe' deve "schedulare" il PSB relativo.

La schedulazione verifica che:

- il PSB sia valido;
- l'applicazione non sia gia' schedulata;
- i data base referenziati siano aperti e abilitati;
- non ci sia conflitto di intenti tra il PSB da schedulare e gli altri già schedulati.

Se l'operazione di schedulazione non va a buon fine, non sara' possibile accedere al data base.

7.5.2 TERMINATING

Quando il programma non ha piu' necessita' immediata di utilizzare i data base descritti nel PSB schedulato, e' bene notificarlo al DL/I rilasciando il PSB.

Questo indica al DL/I che ora lo stesso PSB puo' essere utilizzato da un altro programma e che tutte le modifiche effettuate sui data base, dall'inizio della transazione fino al rilascio del PSB, devono essere considerate definitive e non dovranno essere contro-aggiornate in caso di caduta del sistema.

Inoltre, poiche' in un programma applicativo e' possibile schedulare solo un PSB alla volta, puo' essere necessario rilasciare un PSB per poterne schedulare un altro e, quindi, passare ad elaborare:

- Data Base diversi, non dichiarati nei PCB del vecchio PSB, oppure
- gli stessi Data Base con opzioni di elaborazione (PROCOPT) diverse.

Se non viene rilasciato esplicitamente, il PSB schedulato sara', pero', comunque rilasciato alla fine del task (o ad un "punto di sincronizzazione" del CICS/VS).

7.6 I COMANDI DL/I

I comandi DL/I sono lo strumento, in ambiente CICS/VS, che ci viene messo a disposizione per richiedere i servizi al DL/I.

Questi comandi hanno una sintassi ed un formato simili a quelli CICS/VS (EXEC DLI invece di EXEC CICS) e, all'interno di un programma applicativo, possono essere codificati dovunque necessari.

Essi vengono opportunamente interpretati dal traduttore dei comandi CICS/VS e, in fase di esecuzione, invocano, attraverso il DFHEIP, il DL/I perche' svolga le operazioni richieste.

La sintassi dei comandi DL/I e':

```
EXEC DLI funzione (opzione ((argomento))).....  
END-EXEC
```

Ogni comando EXEC DLI deve includere almeno una funzione e un delimitatore. Molti dei comandi, pero', includeranno anche opzioni ed argomenti.

Un esempio di comando DL/I puo' essere il seguente.

```
EXEC DLI GET UNIQUE USING PCB (1)  
SEGMENT (CLIENTE)  
SEGLENGTH (100)  
WHERE (CODCLI=CODICE)  
FIELDLENGTH (8)  
INTO (IOAREA)  
END-EXEC
```

Questo esempio rappresenta una richiesta di lettura (sul data base indicato nel primo PCB del PSB schedulato) del segmento CLIENTE (lungo 100), avente il suo campo chiave CODCLI (lungo 8) uguale al campo CODICE, descritto e impostato a programma. Il segmento CLIENTE letto deve essere messo nell'area IOAREA definita nel programma.

Vediamo ora di analizzare i componenti di un comando DL/I.

7.7 LA FUNZIONE

Ogni comando deve contenere una funzione che indichi al DL/I l'operazione da intraprendere. Con essa possono essere codificati opzioni ed argomenti necessari per completare l'operazione.

Le funzioni di accesso on-line ai data bases sono:

- GET NEXT abbreviata GN
- GET NEXT IN PARENT abbreviata GNP
- GET UNIQUE abbreviata GU
- INSERT abbreviata ISRT
- REPLACE abbreviata REPL
- DELETE abbreviata DLET
- STATISTICS abbreviata STAT

Nei paragrafi seguenti vedremo l'uso delle opzioni e dei relativi argomenti per queste funzioni di accesso.

Le funzioni di schedulazione e rilascio di PSB in ambiente on-line sono:

- SCHEDULE abbreviata SCHD
- TERMINATE abbreviata TERM

Al paragrafo 7.5 abbiamo visto che, in un programma DL/I online, prima di poter eseguire delle operazioni sui data'bases e' necessario schedulare il PSB che ci permette di accedere ad essi. Il formato del comando e':

```
EXEC DLI SCHD (PSB(nome)/PSB((area-dati))) END-EXEC
```

dove "nome" e' una costante che indica il nome del PSB definito e catalogato dal DBA; mentre "area-dati" e' una variabile opportunamente definita e impostata a programma.

Al paragrafo 7.5 abbiamo anche visto che e' opportuno rilasciare il PSB appena possibile, per sbloccare le risorse e consentirne l'uso alle altre applicazioni. Il formato del comando e' semplicemente:

```
EXEC DLI TERM END-EXEC
```

La descrizione dettagliata di tutte le funzioni sopra elencate si trova nel capitolo 2.3 dell'APRM.

7.8 L'INDICAZIONE DEL PCB

L'opzione PCB specifica un particolare PCB (cioe' una vista di un data base) che deve essere usato per elaborare la funzione richiesta.

Se questa opzione non viene specificata, il DL/I utilizza il primo PCB definito nel PSB schedulato; se, invece, viene codificata, deve seguire subito la funzione.

Il suo formato e': USING PCB (numero) dove "numero" e' un intero o un'espressione COBOL che lo converte ad intero.

7.9 LA SCELTA DEI SEGMENTI

Il segmento finale di una ricerca su un data base, cioè il segmento al più basso livello gerarchico che desideriamo raggiungere, viene indicato come "segmento oggetto". Tutti i segmenti a livelli superiori rispetto al segmento oggetto e posti sullo stesso cammino gerarchico sono detti "segmenti genitori".

In ogni comando di accesso ad un data base (eccetto nei comandi GH e GNP dove e' facoltativo) bisogna specificare al DL/I il segmento oggetto su cui operare ed, eventualmente, uno o più segmenti genitori, che indichino al DL/I il cammino gerarchico per arrivare al segmento oggetto.

7.9.1 Segmenti Genitori

I segmenti genitori, se specificati in un comando di accesso a data base, vanno codificati prima del segmento oggetto e, seguendo l'ordine gerarchico, dal primo livello in poi; senza l'obbligo, però, di doverli elencare tutti.

L'opzione di scelta di un segmento genitore e':

SEGMENT (nome)/SEGMENT ((area-dati))

dove "nome" e' il nome del segmento definito nel blocco DBD; mentre "area-dati" e' il nome di una variabile definita a programma.

7.9.2 Segmenti Oggetti

L'opzione per il segmento oggetto e' identica a quella di un segmento genitore, ma deve essere seguita dall'opzione INTO o FROM (appresso descritte) per individuare l'area di I/O per i dati del segmento. Quindi e' del tipo:

SEGMENT(nome) / SEGMENT((area-dati)) INTO(area)

oppure

SEGMENT(nome) / SEGMENT((area-dati)) FROM(area)

7.9.3 I/O Area di Segmento

L'opzione INTO e' codificata nei comandi di lettura di segmenti (GU, GN e GNP) ed indica l'area in cui depositare i dati del segmento letto.

L'opzione FROM e' codificata nei comandi che modificano il data base (ISRT, REPL e DLET) ed indica l'area da cui prelevare i dati per il segmento.

Queste opzioni, come sopra accennato, sono del tipo:

INTO (area)

FROM (area)

dove "area" e' il nome di un'area di lavoro definita nel programma e grande abbastanza da contenere l'intero segmento da trasferire.

E' possibile codificare l'opzione INTO o FROM anche per i segmenti genitori. In questo caso otterremo anche per essi il trasferimento dei relativi dati nelle aree (o dalle aree) specificate in INTO (o FROM).

Percio' i segmenti genitori possono avere, o meno, definite le opzioni INTO o FROM a seconda della necessita' di trasferire i loro dati.

7.9.4 Lunghezza di Segmento

L'opzione SEGLENGTH definisce la lunghezza del segmento specificato con l'opzione SEGMENT.

In un programma COBOL e' obbligatorio, se abbiamo codificato anche l'opzione INTO o FROM.

L'opzione SEGLENGTH e':

SEGLENGTH (numero)

dove "numero" e' un intero oppure un campo, dichiarato a programma come mezza voce binario, che contenga un intero.

7.9.5 Selezione di Segmenti

Le informazioni necessarie al DL/I per ricercare una particolare ricorrenza di un segmento vengono fornite nell'opzione WHERE.

Quest'opzione va codificata dopo l'opzione SEGMENT cui si riferisce ed e' del tipo:

WHERE (nome oper. rif. (AND/OR nome oper. rif.))
FIELDLENGTH (numero1, numero2,))

dove:

- "nome" e' il nome di un campo definito in DBD nel segmento associato; non puo' essere una variabile.
- "oper." puo' essere uno qualunque dei seguenti comparatori:
 - > maggiore
 - < minore
 - = uguale
 - != diverso
 - <= minore o uguale
 - >= maggiore o uguale
 - -> non maggiore
 - -< non minore
 - =< uguale o minore
 - => uguale o maggiore
- "rif." e' il nome di un campo definito nel programma e contenente l'argomento di ricerca, la cui lunghezza deve essere uguale a quella del campo specificato in "nome".

FIELDLENGTH e' obbligatorio in COBOL ed indica la lunghezza del campo (o dei campi) di ricerca; "numeroN" e' un intero o un campo che contenga un intero definito a programma come mezza voce binaria.

L'operando WHERE puo' contenere espressioni "booleans" (AND e OR), che consentano una identificazione piu' agevole delle ricorrenze di segmento da ricercare.

Per esempio, potremmo chiedere al DL/I di leggere tutte le ricorrenze di un Data Base del Personale che abbiano contemporaneamente le seguenti caratteristiche:

```

ETA' > 18
ETA' < 21
ALTEZZA > 180
SESSO = M
  
```

Nel caso piu' semplice in cui abbiamo specificato solo un argomento di ricerca, l'opzione WHERE opera come segue:

Per ogni ricorrenza del tipo di segmento specificato nell'opzione SEGMENT, il DL/I esamina il campo "nome" e lo confronta con il valore specificato nel campo "rif.", in accordo con l'operatore "oper." codificato. Quando trova un valore che soddisfi la rela-

zione, o quando raggiunge l'ultima ricorrenza del segmento specificato, si ferma.

Nel caso, invece, che l'opzione WHERE abbia codificati piu' argomenti, la ricerca della ricorrenza di segmento e' guidata dalla presenza degli operatori "booleani", per cui il DL/I si ferma solo su un valore che soddisfi tutte le relazioni in AND o una relazione in OR (oppure sull'ultima ricorrenza del segmento specificato).

7.10 RISPOSTE AI COMANDI DL/I

Ogni volta che un programma esegue un comando DL/I, questi gli fornisce una risposta, sull'esito della funzione eseguita, attraverso il blocco DIB (DL/I Interface Block).

Il contenuto di questo blocco e' rappresentato dalle seguenti variabili:

| | Per COBOL |
|----------|----------------|
| DIBSTAT | PIC X(2) |
| DIBSEGM | PIC X(8) |
| DIBSEGLV | PIC X(2) |
| DIBKFBL | PIC S9(4) COMP |

- DIBSTAT e' lo "status code" del DL/I (vedi il paragrafo successivo).
- DIBSEGM e' il nome del segmento letto di piu' basso livello.
- DIBSEGLV e' il livello gerarchico del segmento indicato in DIBSEGM.
- DIBKFBL contiene la lunghezza della chiave concatenata del segmento indicato in DIBSEGM. E' disponibile solo se e' stata specificata l'opzione KEYFEEDBACK nelle funzioni GU, GN o GNP.

Le dichiarative per queste variabili sono generate automaticamente nel programma dal traduttore dei comandi.

La variabile piu' importante e' sicuramente DIBSTAT, che contiene lo "status code" del DL/I relativo all'esecuzione del comando.

7.10.1 Status Code

Dopo l'elaborazione di un comando DL/I, il controllo torna al programma all'istruzione subito seguente il comando stesso. Così e' possibile controllare, attraverso il campo DIBSTAT, se il comando sia stato regolarmente completato e se abbia effettuato sui dati l'operazione richiesta.

Percio' la prima cosa da fare nel programma, dopo ogni comando DL/I, e' provare il contenuto di DIBSTAT per i vari "status codes" ed intraprendere l'azione opportuna.

Gli "status codes" che potrebbero essere restituiti in DIBSTAT sono i seguenti:

- bb per tutte le funzioni. Comando eseguito correttamente.
- GA per le funzioni GN e GNP. Cambiamento di segmento ad un livello piu' alto.
- GB per la funzione GN. Fine Data Set.
- GD per la funzione ISRT. Segmento non specificato.
- GE per le funzioni GU, GN, GNP e ISRT. Segmento non trovato.
- GG per le funzioni GU, GN, GNP. Posizionamento all'inizio del Data Base (PROCOPT=GO).
- GK per le funzioni GN e GNP. Cambiamento di segmento allo stesso livello.
- II per la funzione ISRT. Segmento da inserire gia' esistente.
- TG per la funzione TERM. Tentativo di TERMINATE senza un PSB schedulato.

Essi non necessariamente indicano errore, poiche' i risultati del comando possono essere perfettamente validi, anche se i dati elaborati potrebbero non essere quelli attesi. Spetta al programma verificare questi codici e procedere di conseguenza.

Questi "status codes", reperibili in DIBSTAT, costituiscono pero' solo una parte degli "status codes" DL/I.

La lista completa e' reperibile nel manuale IMS/VS per il programmatore applicativo.

Tutti gli altri "status codes", non reperibili in DIBSTAT, indicano errori non ricoverabili che provocano un abend della transazione. In questo caso il codice di abend ha la forma DHXX, dove "XX" e' lo status code DL/I.

7.10.2 Gestione degli Abend

Come abbiamo visto nel paragrafo precedente, gli "status codes" del DL/I sono gestiti in due modi differenti.

Un primo gruppo di codici viene restituito in DIBSTAT ed analizzato, subito dopo ogni comando DL/I, codificando a programma

l'azione correttiva oppure un salto ad una routine generalizzata per la gestione degli errori.

Gli altri codici di errore provocheranno un ABEND del task e, perciò, non saranno facili da controllare a programma.

E' possibile comunque intercettare l'abend, provocato dall'esecuzione di un comando DL/I, codificando all'inizio del programma il comando EXEC CICS HANDLE ABEND.

Una Exit-Routine di Abend puo' quindi tentare di correggere l'errore e di far proseguire l'elaborazione.

7.11 SOMMARIO

In questo capitolo abbiamo esaminato le caratteristiche delle funzioni DL/I on-line ed i comandi DL/I (simili a quelli CICS/VS) per ottenere i servizi disponibili.

Abbiamo visto che prima di accedere ad un Data Base bisogna lanciare il comando di "scadenzazione" (SCHD) del PSB opportuno e che è bene rilasciarlo (TERM) appena possibile.

Ogni comando DL/I deve indicare la "funzione" richiesta e specificare una o piu' "opzioni", secondo le necessita'.

Soltanente (eccetto GN e GNP) bisognerà indicare il segmento oggetto, quale segmento finale di una ricerca ed, eventualmente, uno o piu' segmenti genitori per facilitare la ricerca al DL/I.

Oltre alle opzioni di tipo SEGMENT ci sarà la necessita' di specificare delle aree di I/O per i segmenti (opzioni INTO o FROM) e le relative lunghezze (opzione SEGLENGTH).

Un'attenzione particolare merita l'opzione WHERE che, facendo uso degli argomenti specificati, permette di selezionare le ricorrenze di segmento da ricercare.

Infine abbiamo visto come utilizzare il blocco DIB e, soprattutto, il campo DIBSTAT, che contiene i codici di ritorno restituiti al programma dopo l'esecuzione di ogni comando DL/I.

adattamento migliora con le altre macchine portatili mentre
l'uso delle stesse altra

in testi che sono trasferiti dalla macchina del testo alla macchina
elettronica si risparmia molto tempo e spazio.

Questo è stato dimostrato da un esperimento condotto dall'Istituto di
Ingegneria dell'università di Roma che ha dimostrato che la scrittura
elettronica è più veloce della scrittura manuale.

Questo è stato dimostrato da un esperimento condotto dall'Istituto di
Ingegneria dell'università di Roma che ha dimostrato che la scrittura
elettronica è più veloce della scrittura manuale.

Altri esperimenti hanno mostrato che la scrittura elettronica è
più veloce della scrittura manuale. Un esperimento condotto dall'Istituto di
Ingegneria dell'università di Roma ha dimostrato che la scrittura elettronica è
più veloce della scrittura manuale.

Un altro esperimento condotto dall'Istituto di Ingegneria dell'università di Roma
ha dimostrato che la scrittura elettronica è più veloce della scrittura manuale.

Un altro esperimento condotto dall'Istituto di Ingegneria dell'università di Roma
ha dimostrato che la scrittura elettronica è più veloce della scrittura manuale.

Un altro esperimento condotto dall'Istituto di Ingegneria dell'università di Roma
ha dimostrato che la scrittura elettronica è più veloce della scrittura manuale.

Un altro esperimento condotto dall'Istituto di Ingegneria dell'università di Roma
ha dimostrato che la scrittura elettronica è più veloce della scrittura manuale.

Un altro esperimento condotto dall'Istituto di Ingegneria dell'università di Roma
ha dimostrato che la scrittura elettronica è più veloce della scrittura manuale.

Un altro esperimento condotto dall'Istituto di Ingegneria dell'università di Roma
ha dimostrato che la scrittura elettronica è più veloce della scrittura manuale.

LC262 - PROGRAMMAZIONE IN AMBIENTE CICS - COBOL

8.1 INTRODUZIONE

Nella prima parte di questo capitolo vedremo come elaborare i data sets VSAM in ambiente CICS/VС e, in particolare, quali comandi del File Control Program (FCP) utilizzare per leggere, scrivere, modificare o cancellare record.

Nella seconda parte vedremo come elaborare sequenzialmente i file VSAM e come utilizzare eventuali indici alternativi.

PARTE PRIMA

8.2 IL FCP

Ricordiamo che il FCP supporta tutte e tre le organizzazioni VSAM (KSDS, ESDS e RRDS) e che ogni cluster, prima di poter essere utilizzato in ambiente CICS/VС, deve essere opportunamente definito dal programmatore di sistema nella File Control Table (FCT).

Il programma applicativo si limiterà ad indicare il nome simbolico dell'entrata in FCT, senza null'altro specificare circa le caratteristiche del data set.

Ricordiamo inoltre che i data sets sono aperti e chiusi automaticamente dal CICS/VС e, perciò, non si devono codificare nei programmi applicativi le istruzioni di OPEN e CLOSE.

8.3 EXCLUSIVE CONTROL DEL VSAM

Una delle funzioni offerte dal FCP è quella di proteggere, attraverso il meccanismo dell'Exclusive Control, i data sets da eventuali aggiornamenti contemporanei sullo stesso record.

In particolare, quando un task legge un record logico VSAM per aggiornamento, il FCP mantiene il controllo esclusivo dell'intero Control Interval che contiene il record, fino a rilascio esplicito.

Vedremo tra breve quali sono i comandi CICS/VС che provocano il rilascio del controllo esclusivo.

Per particolari sulla gestione dei data sets VSAM in ambiente CICS/VС puoi vedere la parte iniziale del Cap. 2.2 dell'APRM.

8.4 DEFINIZIONE DELLE AREE DI I/O

Per elaborare un record di un qualsiasi data set dobbiamo definire un'apposita area di lavoro nel nostro programma.

Questa area, dimensionata sul record logico e non su quello fisico, e' lo spazio in cui mettere un record letto o costruire un record da scrivere.

E' compito dell'Execute Interface Program filtrare le richieste del programma applicativo e muovere i dati tra l'area di lavoro del programma e quella del FCP.

Per esempio, a seguito di un comando di lettura, il FCP legge un intero Control Interval nell'area dati a disposizione del CICS/V5 (FIOA: File I/O Area), ma l'EIP trasferisce nell'area definita dal programma solo il record logico richiesto.

8.5 LETTURA DI UN RECORD

Il comando per leggere un record di un file e':

```
EXEC CICS READ (INTO (nome di campo)/SET (puntatore))
                (LENGTH (nome di campo))
                FILE (nome)
                RIDFLD (nome di campo)
                (KEYLENGTH (numero) (GENERIC))
                (RBA/RRH)
                (GTEQ/EQUAL)
                (UPDATE)
```

Il significato dei parametri e' il seguente:

INTO (nome di campo)

specificava il campo utente entro cui doveva essere masso il record letto.

SET (puntatore)

con questa opzione si chiedeva di leggere il record richiesto direttamente nella FIOA, che e' un'area esterna al programma.

L'argomento e' un "puntatore" che indirizza tale area (vedi il cap.13).

LENGTH (nome di campo)

e' richiesto per elaborare record a lunghezza variabile. Deve essere una mezza voce binaria, impostata con la lunghezza del record piu' lungo.

Dopo la lettura il campo indicato in LENGTH conterrà la lunghezza reale del record letto.

Se la lunghezza del record da leggere e' maggiore di quanto specificato in LENGTH, il record sara' troncato e poi messo nell'area di lavoro (campo INTO). In questo caso sorge la condizione anomala di LENGTH.

FILE (nome)

specifica il nome simbolico (non piu' di 8 caratteri tra spazi) del file che deve coincidere con quello definito in FCT.

RIDFLD (nome di campo)

specifica il nome di un campo utente contenente l'identificatore del record da cercare, cioè la chiave, l'RBA o il RRH del record.

Questo campo, in funzione del tipo di organizzazione, puo' avere formato e contenuto diversi.

- Per files VSAM KSDS, acceduti per chiave, contiene la chiave (intera o generica) del cluster ed ha il formato corrispondente a quello della chiave. Dopo una lettura, il campo di RIDFLD conterrà la chiave intera del record letto; perciò questo deve essere dimensionato sulla chiave intera e mai su quella generica.
- Per files VSAM KSDS, acceduti per RBA, o per files VSAM ESDS, il campo RIDFLD è una voce binaria e contiene i quattro bytes dell'RBA. Se usiamo un "Relative Byte Address", dobbiamo specificare anche l'opzione RBA.
- Per files VSAM RRDS, il campo RIDFLD è una voce binaria e contiene i quattro bytes del Numero Relativo di Record (RRN). Se usiamo un "Numero Relativo di Record", dobbiamo specificare anche l'opzione RRN.

KEYLENGTH (numero) (GENERIC)

specifica la lunghezza della chiave indicata nell'opzione RIDFLD. È obbligatorio codificare questa opzione (insieme all'opzione GENERIC) solo quando si usa una chiave generica di lettura. In tal caso KEYLENGTH deve indicare la lunghezza della chiave generica.

GTEQ/EQUAL

solamente per VSAM KSDS:

GTEQ specifica che se non si trova un record con chiave (completa o parziale) coincidente con quella specificata in RIDFLD, verrà ricercato il record con chiave piu' grande.

EQUAL è il default. Specifica che deve essere ricercato solo il record con chiave (completa o parziale) uguale a quella definita in RIDFLD.

UPDATE

specificava che un record deve essere letto per essere aggiornato o cancellato. Vedi i paragrafi successivi.

8.6 AGGIORNAMENTO DI UN RECORD

Per aggiornare un record bisogna prima leggerlo (comando READ con l'opzione UPDATE), modificarlo in memoria e poi riscriverlo nello stesso posto sul disco.

Per il comando di lettura READ vale tutto quanto abbiamo detto al paragrafo precedente.

Dopo l'esecuzione del comando READ con l'opzione UPDATE, il CICS/V5 manterrà l'"exclusive control" dell'intero Control Interval contenente il record letto.

Il programma applicativo effettuerà gli aggiornamenti previsti nell'area di lavoro (definita a programma) in cui è stato posto il record letto, quindi eseguirà il comando di riscrittura (REWRITE):

```
EXEC CICS REWRITE FILE (nome)
    FROM (nome di campo)
    [LENGTH (numero)]
```

dove:

FILE e' il nome del file (lo stesso del comando di READ con UPDATE);

FROM e' l'area contenente il record da riscrivere;

LENGTH e' la sua lunghezza. Notare che puo' essere un numero, invece che un nome di campo.

Appena eseguito l'aggiornamento, l'"exclusive control" verrà rilasciato e, con esso, tutte le aree di I/O acquisite dal FCP.

Volendo peraltro rilasciare l'exclusive control, senza eseguire l'aggiornamento, occorrerà codificare il comando di UNLOCK. Esempio:

```
EXEC CICS UNLOCK FILE ('FILE')END-EXEC
```

8.7 CANCELLAZIONE DI RECORD

La cancellazione di record è consentita su files VSAM KSDS e RRDS. Poiché spesso è opportuno leggere il contenuto di un record, prima di decidere se cancellarlo o meno, il CICS/V5 offre due possibilità:

- la cancellazione diretta (comando DELETE);
- la cancellazione subordinata (comando READ con opzione UPDATE, seguito dal comando DELETE).

Il formato del comando DELETE è:

```
EXEC CICS DELETE (RIDFLD (nome di campo))
                  (KEYLENGTH (numero))
                  (RBA/RRN)
                  FILE (nome)
                  (GENERIC (NUMREC (nome di campo)))
```

Il record da cancellare puo' essere localizzato tramite chiave, indirizzo relativo di byte (RBA), o numero relativo di record (RRN).

Possono anche essere cancellati gruppi di record, ma solo se hanno in comune un certo insieme di caratteri della chiave (cioe' con stessa chiave parziale).

RIDFLD (nome di campo)
obbligatorio con GENERIC.
Valgono le stesse considerazioni viste per il comando READ. In piu' bisogna notare che questa opzione deve essere omessa se il comando DELETE segue un comando READ con UPDATE e solo nel caso che non sia usata la ricerca con chiave parziale.

(KEYLENGTH (numero))
quando si usa una chiave parziale si deve codificare questa opzione specificando la lunghezza della chiave parziale.

GENERIC
indica che si usa una chiave parziale che identifica un gruppo di record da cancellare.

NUMREC (nome di campo)
e' usata sempre con GENERIC e fa riferimento ad un campo binario lungo due byte che contiene il numero di record VSAM da cancellare.

Da quanto sopra risulta che esistono piu' modi per rilasciare l'exclusive control innescato da un comando di READ UPDATE; precisamente:

- con un comando REWRITE
- con un comando UNLOCK
- con un comando DELETE
- a chiusura del task.

8.8 SCRITTURA DI NUOVI RECORD

Il comando per aggiungere record ad un file e' il seguente:

```
EXEC CICS WRITE FROM (nome di campo)
                  (LENGTH (numero))
```

FILE (nome)
RIDFLD (nome di campo)
(KEYLENGTH (numero)
(RBA/RRN)
(MASSINSERT)

Per quasi tutte le opzioni del comando WRITE valgono le considerazioni fatte per le corrispondenti opzioni del comando READ. Le uniche opzioni nuove sono:

FROM (nome di campo)
specifica il campo in cui il programma applicativo ha posto il record da scrivere sul file.

MASSINSERT
specifica che la richiesta di WRITE in oggetto fa parte di una operazione di inserimento di massa. Una tale operazione deve essere chiusa, tramite il comando UNLOCK oppure a chiusura del task; solo a chiusura dell'operazione i record verranno fisicamente registrati su data set.

Notare che l'opzione LENGTH puo' essere una costante.

8.9 LE CONDIZIONI ANOMALE

Le condizioni anomale che possono verificarsi durante le operazioni di FCP fin qui esaminate son raggruppabili in 3 sezioni:

1.

FILE NOT errore sulla identificazione del DATASET
FOUND

INVREQ richiesta invalida

ILLOGIC richiesta logica

2.

NOTFND record di lettura non trovato

DUPREC record duplicato (nel caso si voglia aggiungere un record)

DUPKEY chiave duplicata (solo quando si usano gli indici alternativi in VSAM)

LENGERR il record cercato con una operazione di READ eccede in lunghezza l'area di programma predisposta a riceverlo.

3.

IOERR errore di I/O

NOSPACE mancanza di spazio a seguito di una WRITE

NOTOPEN il data set richiesto non e' stato aperto.

Per gestire queste condizioni anomale si puo' usare il comando HANDLE CONDITION.

Il sistema, per default, chiude il task in maniera anomala. Il consiglio e':

per la sezione 1. lasciare terminare il task in maniera anomala (azione di default)

per la sezione 2. gestire con un comando HANDLE CONDITION il verificarsi di una o piu' di queste condizioni

per la sezione 3. mandare un messaggio all'operatore e poi eventualmente far chiudere in modo anomale il task.

E possibile reperire una lista completa di tutte queste condizioni anomale nel cap. 2.4 dell'APRM.

PARTE SECONDA

8.10 BROWSING

Nella prima parte di questo capitolo abbiamo visto quali sono i comandi CICS/V5 per leggere, scrivere, cancellare e aggiornare singoli record su files VSAM.

Ora vedremo come elaborare sequenzialmente questi files. In ambiente CICS/V5 l'elaborazione sequenziale di record è detta browsing.

Quanti hanno familiarità con la gestione batch dei file sanno che, qualora un archivio sia organizzato in modo da rendere possibile l'accesso diretto ai dati, la sua lettura sequenziale richiede l'uso di istruzioni particolari, diverse da quelle utilizzate normalmente per l'accesso al singolo record. La stessa situazione si presenta per chi programma utilizzando i comandi CICS/V5, ma le difficoltà di programmazione sono qui molto minori.

Sono possibili inoltre, all'interno dello stesso programma, scansioni di porzioni diverse dello stesso file e scansioni di file diversi: l'operazione prende il nome di "browsing multiplo".

8.11 ELEMENTI DI UN BROWSING

Una scansione sequenziale consta di quattro fasi:

1. posizionamento iniziale su di un determinato record
2. ciclo di lettura sequenziale di un certo numero di record a discrezione dell'utente
3. eventuale riposizionamento per proseguire la ricerca su altra parte del file o per modificare la chiave di ricerca
4. chiusura del browsing.

Nella fase di posizionamento (comando STARTBR) occorre:

- specificare il nome del file cui si intende accedere
- fornire un campo contenente l'argomento di ricerca del record da cui iniziare la scansione
- attribuire un codice specifico ed univoco alla operazione che ci si appresta a iniziare per mantenerla distinta da altre dello stesso tipo.

E' possibile inoltre specificare le modalità di posizionamento, che può essere sul record con chiave uguale, o maggiore-uguale di quella fornita.

L'argomento di ricerca puo' essere:

- RBA (4 byte binari)
- RRN (4 byte binari)
- chiave intiera
- chiave generica (o parziale)

Vedremo che la lettura di un file VSAM puo' essere effettuata sia in avanti (chiavi crescenti) sia all'indietro (chiavi decrescenti) con i comandi READNEXT e, rispettivamente, READPREV.

In fase di lettura, ad ogni comando il CICS/V5 mette a disposizione del programma applicativo, via via che vengono richiesti, i record logici e la chiave relativa, ma non sono consentite operazioni di aggiornamento.

Per variare un record occorre rileggerlo, con un comando di READ con opzione UPDATE, utilizzando la chiave (o RBA o RRN), restituita dal CICS/V5 al programma, nello stesso campo usato per fornire l'argomento di ricerca del posizionamento, cioè nel campo dell'opzione RIDFLD.

Quindi sara' possibile modificare il record e poi riscriverlo. L'operazione di aggiornamento non chiude il browsing, che puo' proseguire pertanto senza problemi.

Nota pero' che questa tecnica puo' provocare un "deadlock" in ambiente di Risorse VSAM Condivise, che puo' essere evitato chiudendo tutti i browsing su quel data set, prima di eseguire un aggiornamento. Per questo argomento puoi consultare l'apposito paragrafo del cap.2.2 dell'APRM.

A volte ci puo' essere la necessita' applicativa di elaborare sequenzialmente solo porzioni limitate del file, e non tutti i record in esso contenuti.

Per evitare di chiudere il browsing corrente ed iniziare uno nuovo in posizione diversa, il CICS/V5 ci mette a disposizione un comando di riposizionamento (comando RESETBR) all'interno del file, che non provoca la chiusura del browsing.

Inoltre, il comando di riposizionamento puo' essere utilizzato anche per cambiare il tipo di ricerca in un file VSAM KSDS (per esempio da chiave intiera a chiave generica).

La fase di chiusura (comando ENDBR) consiste nel segnalare al CICS/V5 che l'operazione di browsing e' terminata, e che i blocchi di controllo e le risorse (aree di I/O e work-area) impiegate per la sua gestione possono essere rilasciate.

8.12 COMANDI DI BROWSING

8.12.1 Posizionamento

```
EXEC CICS STARTBR FILE (nome)
                   RIDFLD (nome di campo)
                   (KEYLENGTH (numero))(GENERIC))
                   (GTEQ/EQUAL)
                   (RBA/RRH)
                   (REQID (numero))
```

Questo comando e' usato per specificare il record da cui deve iniziare il browsing. Il comando di per se' non fornisce al programma applicativo alcun record; l'evento si realizza invece all'esecuzione del comando READNEXT o READPREV.

Per posizionare il browsing all'inizio del file, possiamo specificare:

- un RIDFLD di zeri esadecimali, oppure
- le opzioni KEYLENGTH(0) e GENERIC

Per posizionare, invece, il browsing alla fine del file (per lettura all'indietro), possiamo specificare un RIDFLD di 'FF' esadecimale.

Il significato dei parametri e' il seguente:

FILE (nome)

specifica il nome del file su cui esegue la ricerca sequenziale. Non piu' di 8 caratteri tra apici, identici al nome riportato nella FCT.

RIDFLD (nome di campo)

indica il nome simbolico del campo nel quale l'utente ha preparato la chiave del record su cui il CICS/V5 si deve posizionare. La chiave puo' essere:

- Per VSAM KSDS parziale o completa, ma in ogni caso il campo che la contiene deve essere definito di lunghezza pari a quella della chiave completa. Se si usa chiave parziale, i caratteri che la compongono vanno allineati a sinistra nel campo chiave. Con una chiave parziale e' obbligatorio codificare anche KEYLENGTH e GENERIC.
- Per VSAM ESDS il campo contiene un RBA. L'opzione RBA diventa obbligatoria.
- Per VSAM RRDS il campo contiene un RRN. Deve essere codificata anche l'opzione RRN.

GENERIC

indica l'uso di chiave parziale e richiede la codifica del parametro KEYLENGTH.

KEYLENGTH (numero)

se e' stato specificato GENERIC, questo parametro indica la lunghezza della chiave parziale; in altre parole, il numero di caratteri, a partire da sinistra, che compongono, all'interno del campo chiave, la chiave parziale. Se GENERIC non e' stato codificato, KEYLENGTH non e' obbligatorio e deve indicare la lunghezza della chiave completa, come scritto in FCT.

GTEQ/EQUAL

GTEQ e' il default. Indica che il posizionamento avverra' sul primo record la cui chiave (totale o parziale) verrà trovata uguale o maggiore di quella contenuta nel campo RIDFLD.

EQUAL indica che il posizionamento avverra' sul record la cui chiave (totale o parziale) e' uguale a quella contenuta nel campo RIDFLD.

RBA/RRN

RBA indica che il contenuto di RIDFLD e' un Relative Byte Address (4 byte binari).

RRN che tale contenuto e' un Relative Record Number (4 byte binari).

REQID (numero)

serves ad attribuire un numero di identificazione che contraddistingua in maniera unica una richiesta di browsing, utilizzabile in operazioni di browsing multiplo. Se omesso, viene assunto come numero della richiesta zero.

8.12.2 Lettura Sequenziale

```
EXEC CICS READNEXT (INTO (nome di campo)
                     / SET (puntatore))
                     (LENGTH (nome di campo))
                     FILE (nome)
                     RIDFLD (nome di campo)
                     (KEYLENGTH (numero))
                     (RBA/RRN)
                     (REQID (numero))
```

Con questo comando si ottiene l'accesso al record logico nel campo specificato col parametro INTO, oppure in un'area gestita dal CICS/V5 il cui indirizzo viene posto nel "puntatore" il cui nome compare come argomento di SET. (Vedi il cap. 14).

Il campo chiave RIDFLD va sempre indicato e deve essere lo stesso specificato nel comando STARTBR; in esso viene restituita al programma applicativo la chiave completa del record letto. La lun-

ghezza della chiave deve essere, se specificata, la stessa codificata nella FCT per la chiave completa. Il significato dei parametri e' quello visto nel paragrafo precedente e in quello che tratta del comando READ per i file.

In ambiente VSAM e' possibile eseguire anche letture sequenziali all'indietro, utilizzando il comando:

```
EXEC CICS READPREV (INTO (nome di campo)
                      / SET (puntatore))
                      (LENGTH (nome di campo))
                      FILE (nome)
                      RIDFLD (nome di campo)
                      (KEYLENGTH (numero))
                      (RBA/RRN)
                      (REQID (numero))
```

Il significato e l'uso dei parametri sono i medesimi visti per il comando READNEXT, con le seguenti differenze:

- In lettura all'indietro non e' possibile specificare una chiave generica. Se e' specificata scatta la condizione anomala INVREQ (richiesta invalida).
- Il record specificato nel comando STARTBR deve esistere sul file, altrimenti al primo comando READPREV scatta la condizione NOTFND (non trovato).

Nota che un comando READPREV, successivo ad un comando READNEXT, ottiene la rilettura dello stesso record.

Una lettura sequenziale, con loop di READNEXT o di READPREV, se non e' interrotta da una richiesta esplicita del programma o dal verificarsi di una situazione di errore particolare, andra' avanti fino alla fine del file.

Dopo che l'ultimo record del file e' stato letto regolarmente, un successivo comando di lettura fa scattare la condizione anomala di ENDFILE (con un loop di READPREV la condizione di ENDFILE si verifica sul primo record del file).

E' bene che ogni programma includa una HANDLE CONDITION per la condizione ENDFILE, indipendentemente dal fatto che si voglia o meno leggere tutto il file.

8.12.3 Riposizionamento

Si supponga di aver eseguito una ricerca sequenziale su di un file con chiave di partenza 1000 e di aver letto n record. Se si desidera continuare la ricerca su altra parte dello stesso file, e' possibile eseguire l'operazione di riposizionamento evitando di chiudere il browsing e di riaprirne un altro.

Il comando che consente di realizzare l'operazione e':

```
EXEC CICS RESETBR FILE (nome)
    RIDFLD (nome di campo)
        (KEYLENG1H (numero)(GENERIC))
        (GTEQ/EQUAL)
        (RBA/RRN)
        (REQID (numero))
```

I parametri hanno lo stesso significato di quelli elencati nel paragrafo relativo al comando di posizionamento. Ma in particolare RIDFLD deve indicare la nuova posizione all'interno del file da cui intendiamo riprendere il browsing. Questa nuova posizione deve essere impostata in RIDFLD prima di inviare il comando RESETBR.

Il nome di campo dell'opzione RIDFLD deve indicare lo stesso campo codificato nel comando STARTBR.

Durante un browsing, con questo comando, oltre che modificare la posizione all'interno del file, possiamo anche cambiare:

- il tipo di chiave di ricerca (generica o intiera) specificata nello STARTBR;
- il tipo di ricerca (per GTEQ o per EQUAL).

8.12.4 Chiusura

```
EXEC CICS ENDBR FILE (nome)
    (REQID (numero))
```

I parametri hanno i significati ormai noti. Se viene omesso il comando ENDBR, l'operazione di browsing viene chiusa dal CICS al termine del task; ricordiamo peraltro che e' sempre opportuno chiudere esplicitamente la lettura sequenziale per liberare al piu' presto le risorse di sistema (buffer e blocchi di controllo) altrimenti bloccate fino al termine del task.

8.13 ELABORAZIONE SKIP-SEQUENTIAL

E' una tecnica VSAM, estesa in ambiente CICS/VS, che permette di saltare in avanti su un file modificando il contenuto del campo contenente l'argomento di ricerca, cioe' il RIDFLD. Questo metodo permette un accesso "random" veloce sul file VSAM riducendo il tempo di ricerca nell'indice.

Cioe' significa che, nel caso dobbiamo effettuare un riposizionamento in avanti su un file, durante un browsing, non e' necessario eseguire il comando RESETBR, a patto, pero', che non modifichiamo:

- l'argomento di ricerca (chiave, RBA o RRN)
- il tipo di ricerca (chiave generica o intiera)

- la modalita' di ricerca (per chiave maggiore o maggiore-uguale).

L'elaborazione skip-sequential e' realizzata semplicemente impostando l'argomento di ricerca del nuovo record da leggere nel campo specificato per l'opzione RIDFLD. Al successivo comando di lettura (READNEXT), il VSAM riconosce la modifica di sequenza e si posiziona sul file in accordo con essa.

8.14 BROWSING MULTIPLO

In un singolo programma applicativo e' possibile eseguire piu' operazioni di browsing contemporanea su files diversi o sullo stesso file.

Quando il browsing multiplo viene eseguito sullo stesso file, e' necessario distinguere tra loro le varie operazioni di browsing assegnando a ciascuno un numero univoco che lo contraddistingua.

Questo e' realizzato dando un valore all'opzione REQID nel comando STARTBR.

Tale valore, poi, va sempre citato nell'opzione REQID per ogni comando di lettura, di riposizionamento e di chiusura.

Quando il browsing multiplo viene eseguito su piu' files, l'opzione REQID non e' piu' necessaria.

8.15 INDICI ALTERNATIVI VSAM

Costruire un indice alternativo su un file VSAM significa permettere l'accesso ai record del data set attraverso una chiave alternativa rispetto a quella inizialmente esistente.

Il file su cui viene costruito l'indice alternativo e' detto "cluster base". L'indice alternativo e il suo cluster base sono data set fisici distinti, ma collegati logicamente da un "path". Nella FCT sono definiti, oltre ai cluster base, anche i path. Quindi per elaborare un cluster base attraverso un suo indice alternativo basta indicare il nome del path relativo.

L'elaborazione di un record attraverso un indice alternativo e' uguale a quella di un normale data set KSDS, a meno che siano coinvolti records con chiave alternativa non-unica.

In questo caso, a seguito di un comando di lettura (READ, READNEXT o READPREV), viene letto il primo record nel data set con chiave non-unica, ma scatta la condizione di "record duplicato" (DUPKEY).

L'azione di default del CICS/V5 sarebbe quella di chiudere abnormalmente il task; perciò, per continuare l'elaborazione, e'

necessario codificare un comando HANDLE CONDITION con l'opzione DUPKEY.

Un modo per leggere gli altri record, aventi la stessa chiave, e' quello di iniziare un browsing e continuare con dei comandi di READNEXT (o READPREV).

La condizione di DUPKEY si ripete ad ogni READNEXT (o READPREV) che legga una chiave non unica, eccetto l'ultima. Per esempio, se ci sono tre record con la stessa chiave alternativa, la condizione DUPKEY scatta per i primi due record, ma non per il terzo.

8.16 CONSIDERAZIONI SULL'USO DEL BROWSING

E' importante ricordare che, per eseguire le transazioni CICS/V5, abbiamo a disposizione una quantita' definita di risorse, il cui utilizzo inefficiente da parte anche di una sola transazione puo' influenzare la risposta complessiva dell'intero sistema, specialmente se tale transazione viene eseguita contemporaneamente da piu' terminali.

Quando un operatore immette una richiesta per leggere dati, desidera che la risposta gli sia fornita subito. Se il programma prevede un comando di posizionamento sul file e, poi, dei comandi di lettura sequenziale dei record, nel caso piu' favorevole il tempo di reperimento del record richiesto sara' relativamente breve (pochi secondi), ma pur sempre lungo per l'operatore in attesa. E' importante, perciò, limitare la durata di un browsing restringendo l'intervallo di ricerca.

L'obiettivo da raggiungere nello scrivere transazioni di browsing e' quello di minimizzare l'uso della memoria. Questo puo' essere fatto usando l'opzione SET nei comandi di READNEXT e READPREV. Infatti, in questo modo e' possibile leggere i record direttamente nel buffer del CICS/V5, senza bisogno di muoverli nell'area di memoria del programma, risparmiando cosi' nella dimensione del programma stesso. Con l'elaborazione nel buffer del CICS/V5 si ha anche il vantaggio di risparmiare il tempo necessario per spostare i dati dal buffer all'area di lavoro del programma.

Sebbene le operazioni di browsing possano essere eseguite per leggere e visualizzare anche singoli record, questa tecnica e' più comunemente usata per visualizzare gruppi di record opportunamente accumulati (vedi il PAGING del cap. 13).

8.17 LE CONDIZIONI ANOMALE DI BROWSING

Durante un browsing possono verificarsi le seguenti condizioni di errore:

1. STARTBR

- DSIDERR nome del file errato (FILE NOT FOUND)
- NOTOPEN file non aperto
- NOTFND la chiave specificata non esiste
- ILLOGIC errori VSAM che non rientrano nella casistica presente, la cui gestione richiede l'esame dei Return Code nel EIB
- INVREQ il REQID specificato esiste già per un altro browsing all'interno dello stesso task
- IOERR errore di input/output

2. READNEXT/READPREV

- DSIDERR come sopra
- NOTOPEN come sopra
- NOTFND come sopra relativo a SKIP-SEQUENTIAL
- ILLOGIC come sopra
- IOERR come sopra
- INVREQ il comando è relativo ad un file per cui non esiste precedente STARTBR; oppure la lunghezza della chiave non è congruente con la FCT e non è stato specificato GENERIC, oppure è stato specificato GENERIC ma KEYLENGTH fornisce per la chiave un valore negativo o superiore a quello della FCT; oppure il nome del campo chiave fornito in RIDFLD non si mantiene costante durante il browsing; oppure il metodo di ricerca (RBA, RRN, chiave, ...) non si mantiene costante durante il browsing e viene modificato senza un comando di RESETBR
- LENGERR la lunghezza del record letto eccede quella specificata nel comando
- DUPKEY relativo a indici alternativi VSAM: la chiave secondaria punta a più di una chiave primaria
- ENDFILE è stata raggiunta la fine del file

3. RESETBR

ILLOGIC, INVREQ, NOTFND, NOTOPEN, IOERR

4. ENDBR

ILLOGIC, INVREQ

Per una descrizione completa guarda la fine del cap. 2.4 dell'APRM.

8.18 SOMMARIO

Abbiamo visto come il File Control Program (FCP) utilizzi il metodo d'accesso VSAM per fornire i servizi di elaborazione dei file.

Durante la generazione del sistema CICS/VIS, viene creata una File Control Table (FCT) che definisce il nome e le caratteristiche per ogni file che debba essere elaborato da un programma CICS/VIS.

Il FCP mette a disposizione del programma applicativo i singoli record logici, che vengono quindi elaborati in un'area di lavoro appositamente definita nel programma.

Una delle principali funzioni del FCP e' di fornire l'"exclusive control" durante l'aggiornamento di un record da parte di un task.

Il programma puo' comunque rilasciare il controllo esclusivo con un comando UNLOCK.

I comandi CICS/VIS per elaborare i file sono READ, WRITE, REWRITE e DELETE.

Speciale attenzione va fatta sull'uso dell'opzione RIDFLD, che indica l'area dove la chiave del record da leggere.

L'elaborazione sequenziale di un file e' detta "browsing". Esso e' costituita di tre fasi:

1. comando STARTBR: stabilisce la posizione nel file da cui cominciare la lettura;
2. comando READNEXT (o READPREV): legge sequenzialmente i record;
3. comando ENDBR: chiude il browsing.

Durante un browsing e' sempre possibile ristabilire la posizione di lettura, facendo uso del comando RESETBR e dell'opzione RIDFLD opportunamente impostata. In casi particolari, e' possibile ristabilire la posizione di lettura impostando solo l'argomento dell'opzione RIDFLD, con la chiave successiva da cui deve continuare il browsing (tecnica "skip sequential").

In un singolo programma sono consentite anche operazioni di browsing multiplo sullo stesso file o su file diversi. In questo caso e' necessario specificare l'opzione REQID per distinguere le diverse operazioni di browsing.

Se usiamo il browsing attraverso un Indice Alternativo VSAM, bisogna gestire l'eventualita' di "chiave duplicata" (DUPKEY) per poter proseguire l'elaborazione sequenziale.

E' importante limitare la durata di un browsing per minimizzare il tempo di risposta all'operatore richiedente.

Inoltre, per evitare che il browsing sottragga excessive risorse al sistema, si puo' far uso dell'opzione SET (nei comandi READNEXT o READPREV), che consente di elaborare i dati direttamente nei buffer del CICS. Questa tecnica ha anche il vantaggio di essere piu' veloce della tecnica "move", che invece muove i dati dal buffer del CICS all'area di lavoro del programma applicativo.

Problema: il settore deve contenere l'indirizzo del record da leggere. Se questo non e' conoscibile, come accade nel caso di un file di log, non e' possibile utilizzare la tecnica "move".

Per soluzioN, utilizziamo l'opzione READNEXT dell'utente che consente di trarre in considerazione, insieme "l'ultimo record letto", l'indirizzo del record successivo. Esso e' determinato dalla formula:

INDRISCO - 1 + 1000 * (NREC - 1) + 1000 * (NREC - 1) + 1000 * (NREC - 1)

che, applicata alle varie righe, consente di evitare eventuali errori.

Per esempio, se l'utente ha scritto l'indirizzo del record precedente, l'utente deve indicare l'indirizzo del record successivo.

Per esempio, se l'utente ha scritto l'indirizzo del record precedente, l'utente deve indicare l'indirizzo del record successivo.

Per esempio, se l'utente ha scritto l'indirizzo del record precedente, l'utente deve indicare l'indirizzo del record successivo.

Per esempio, se l'utente ha scritto l'indirizzo del record precedente, l'utente deve indicare l'indirizzo del record successivo.

Per esempio, se l'utente ha scritto l'indirizzo del record precedente, l'utente deve indicare l'indirizzo del record successivo.

9 I DATA BASE DB2

9.1 INTRODUZIONE

In questo capitolo vedremo come è possibile trattare in ambiente on-line i dati memorizzati in un Data Base DB2.

In particolare ci soffermeremo su quelle istruzioni che ci permettono di accedere e di manipolare i dati.

9.2 DEFINIZIONE DI UN DATA BASE DB2

DB2 utilizza il modello relazionale dei dati. Ogni RELAZIONE viene rappresentata all'interno di una riga. In DB2, tutti i dati, vengono definiti, utilizzati e presentati come delle tabelle bidimensionali costituite da righe (in senso orizzontale) e da colonne (in senso verticale).

9.3 STRUCTURED QUERY LANGUAGE (SQL)

L'SQL è il linguaggio evoluto per la gestione dei dati in DB2.

Questo linguaggio consente non solo di accedere ai dati contenuti nel DB2 e di elaborarli (Data Manipulation Language), ma consente inoltre la loro definizione (Data Definition Language) e di gestire la sicurezza e la riservatezza dei medesimi tramite la concessione di autorizzazioni (Data Control Language).

9.4 DEFINIZIONE DELLE AREE DI I/O

Per richiedere servizi al DB2 bisogna definire all'interno del programma applicativo delle aree di input-output necessarie a fornire e ricevere informazioni dall'interfaccia del DB2.

Queste aree sono:

- **HOST VARIABLES:** sono le variabili definite all'interno del programma applicativo destinate a "CONTENERE" le informazioni "DA" e "VERSO" il DB2.
- **INDICATOR VARIABLES:** sono indicatori che permettono di gestire il valore NULLO nelle HOST VARIABLES:
Possono assumere i seguenti valori:

| valore indicatore | significato |
|----------------------|---|
| uguale a zero | la variabile NON contiene un valore NULLO |
| minore di zero | la variabile contiene un valore NULLO |
| maggiore di zero | la variabile contiene un valore troncato |

- SQLCA: in questa area, definita all'interno del programma applicativo (exec sql include SQLCA), vengono posti i codici di ritorno, forniti da SQL, ad ogni richiesta di servizio fatta a DB2;

struttura della SQLCA:

| variabile | significato |
|-----------|-------------------------------------|
| sqlcaid | char (08) identificativo |
| sqlcabc | bin fixed (31) lunghezza (136) |
| SQLCODE | bin fixed (31) RETURN CODE |
| sqlerrm | char (70) messaggio errore |
| sqlerrp | char (08) proc. in errore |
| sqlerrd1 | bin fixed (31) diagn.eccez.-rds |
| sqlerrd2 | bin fixed (31) diagn.eccez.-d.m |
| sqlerrd3 | bin fixed (31) num.righe I.D.U |
| sqlerrd4 | bin fixed (31) riservato |
| sqlerrd5 | bin fixed (31) colonna in err. |
| sqlerrd6 | bin fixed (31) err-code buf mgr |
| sqlwarn0 | char (01) riservato |
| sqlwarn1 | char (01) truncamento |
| sqlwarn2 | char (01) nulls in funz. di calcolo |
| sqlwarn3 | char (01) host vars < cols |
| sqlwarn4 | char (01) no where in DEL / UP |
| sqlwarn5 | char (01) stmt sql/ds |
| sqlwarn6 | char (01) riservato |
| sqlwarn7 | char (01) riservato |
| sqlext | char (08) riservato |

9.5 UNIT OF RECOVERY

UNIT OF RECOVERY: è una sequenza di accessi a DB2 riferiti ad una o più tabelle che deve essere sviluppata in modo completo affinché non venga meno la consistenza dei dati.

Una UOR inizia implicitamente all'interno di un programma quando viene eseguito il primo STATEMENT SQL.

Una UOR in ambiente CICS termina con lo statement EXEC CICS SYNCPOINT (convalidando gli aggiornamenti) oppure EXEC CICS SYNCPOINTROLLBACK (non convalidando gli aggiornamenti).

Una UOR termina, in modo implicito, con la fine del programma; se il programma termina correttamente gli aggiornamenti vengono convalidati; se il programma va in ABEND gli aggiornamenti NON vengono convalidati.

L'operatività dei programmi può essere segmentata in UOR.

9.6 GLI STATEMENTS SQL

STATEMENTS SQL: sono lo strumento che in ambiente CICS/VС
ci viene messo a disposizione per richiedere i servizi al
DB2. La loro sintassi è molto simile a quella del CICS/VС e
possono essere codificati in qualsivoglia sequenza
all'interno del programma applicativo.

La sintassi degli statements SQL è:

EXEC SQL



END-EXEC

Alcuni esempi di statements SQL:

a)

```
EXEC SQL
    DECLARE xyz CURSOR FOR
        SELECT campo1,
               campo2
        FROM tabella_sel
        WHERE campo1 = :host-ricerca
    END-EXEC
```

L'esempio rappresenta la definizione di un cursore di nome xyz funzionale allo scorrimento delle righe delle tabelle DB2 "tabella_sel" che hanno la colonna campo1 UGUALE al contenuto della variabile "host-ricerca".

b)

```
EXEC SQL
    OPEN xyz
END-EXEC
```

L'esempio rappresenta l'apertura del cursore di nome XYZ.

c)
EXEC SQL
 FETCH xyz
 INTO :host1:
 :host2:ind2
END-EXEC

L'esempio rappresenta la acquisizione delle colonne indirizzate dal cursore XYZ e poste rispettivamente nelle variabili HOST1 e HOST2. impostando inoltre gli indicatori del valore NULLO.

d)
EXEC SQL
 CLOSE xyz
END-EXEC

L'esempio rappresenta la chiusura del cursore di nome XYZ.

e)
EXEC SQL
 DELETE
 FROM tabella_sel
 WHERE campol = :host-ricerca
END-EXEC

L'esempio rappresenta la cancellazione di tutte le righe delle tabelle DB2 "tabella_sel" che hanno la colonna campol UGUALE al contenuto della variabile "host-ricerca".

f)
EXEC SQL
 UPDATE tabella_sel
 SET campol = :host1.
 campo2 = :host2.
 WHERE campol = :host-ricerca
END-EXEC

L'esempio rappresenta l'aggiornamento delle colonne campol e campo2 in tutte le righe delle tabella_sel" che hanno la colonna campol UGUALE al contenuto della variabile "host-ricerca". con il valore presente rispettivamente in HOST1 e HOST2.

9.7 LOCKING DB2

La funzione di locking del DB2 è un meccanismo interno automatico che acquisisce i vincoli (LOCKS) sui dati a cui una UOR accede; tale funzione serve a garantire la integrità logica dei dati contenuti nelle tabelle.

Questo meccanismo si occupa della condivisibilità dei dati, controlla quindi l'accesso da parte di più utenti agli stessi dati ed impedisce l'accesso a dati UNCOMMITTED.

In DB2 vengono gestiti tre tipi di locks:

- EXCLUSIVE Viene impedito ad altri utenti di leggere o modificare i dati. Tutte le UOR acquisiscono automaticamente i locks esclusivi su tutti i dati che vengono modificati al loro interno.
- UPDATE Vengono vincolati i dati che potrebbero essere modificati. Gli altri utenti possono leggere questi dati, ma non possono ottenere un lock di tipo EXCLUSIVE o di tipo UPDATE per modificarli.
- SHARE Viene consentito ad altri utenti di leggere i dati, ma viene impedito loro di modificarli. Le UOR acquisiscono automaticamente un lock di tipo SHARE su dati che leggono.

9.8 RISPOSTE AGLI STATEMENTS SQL

Ogni volta che un programma esegue uno statement SQL, questo fornisce l'esito dello stesso nella struttura della SQLCA. Il contenuto della variabile SOLCODE deve perciò essere controllato dopo ogni statement SQL per intraprendere l'azione opportuna.

9.9 GESTIONE GENERALIZZATA DEGLI ERRORI SQL

Come abbiamo descritto precedentemente, è necessario dopo ogni statement SQL controllare il contenuto della variabile SQLCA: è possibile comunque gestire in modo generalizzato gli errori SQL con gli statements WHENEVER che sono:

- EXEC SQL WHENEVER NOT FOUND
go to non-trovato
END-EXEC
- EXEC SQL WHENEVER SQLWARNING
CONTINUE
END-EXEC
- EXEC SQL WHENEVER SQLERROR
go to errore-sql
END-EXEC

questi statements hanno effetto in fase di precompilazione

9.10 PRECOMPILAZIONE DEL PGM DB2

Gli statements SQL inseriti in un programma devono essere analizzati e convertiti prima della normale compilazione del linguaggio host.

Questa fase di "preprocessing" trasforma il source iniziale in modo che le istruzioni SQL non siano più presenti, sostituendole con una serie di istruzioni di CALL all'interfaccia del DB2. Il source così modificato può essere elaborato dal preprocessore CICS, poi dal compilatore del linguaggio ospite utilizzato ed infine dal linkage editor.

Durante la fase di preprocessing DB2 viene prodotto un DBRM (D.B. Request Module) che contiene le istruzioni SQL inserite nel source del programma. Questo DBRM diviene l'input per la fase di BIND, la quale produce il PLAN utilizzato dal DB2 per accedere ai dati al momento dell'esecuzione del programma.

9.11 SOMMARIO

In questo capitolo abbiamo esaminato come accedere al DB2 utilizzando il linguaggio SQL all'interno dei programmi.

Abbiamo visto quali sono le aree di comunicazione tra il programma ed il DB2; particolare attenzione dovrà essere posta all'utilizzo degli statement SQL e per gli stessi si rimanda al manuale Application Programming DB2.

Infine abbiamo evidenziato quali sono i passi necessari alla preparazione e all'utilizzo di un programma DB2.

10.1 INTRODUZIONE

In questo capitolo vedremo una delle due funzioni offerte dal CICS/V5 per la gestione dei dati temporanei: il "Transient Data Control".

In particolare vedremo quali sono i tipi di "coda" disponibili per scrivere o leggere dati temporanei ed i relativi comandi del Transient Data Control Program (TDP).

10.2 GENERALITÀ

I dati temporanei che devono essere gestiti dal TDP sono memorizzati su data sets sequenziali noti col nome di "coda" o "destinazioni".

Il TD e' quindi una tecnica generalizzata di accodamento dei dati che consente di scrivere su (o leggere da) una predefinita coda o destinazione, utilizzando il nome simbolico che e' servito a definirne l'entrata nella Destination Control Table (DCT).

Le destinazioni di Transient Data possono essere definite come:

- **Extrapartition**
- **Intrapartition**
- **Indirette**

E' compito del programmatore di sistema definire in DCT il tipo delle destinazioni e assegnare i nomi simbolici.

10.3 DESTINAZIONI EXTRAPARTITION

Le destinazioni extrapartition sono code di dati temporanei create da un task CICS/V5 ed elaborate fuori della regione CICS/V5, o viceversa.

Ogni destinazione e' un file sequenziale risiedente su un qualsiasi dispositivo sequenziale (DASD, nastro, stampante) ed e' definita nella DCT come file di input o di output rispetto alla regione CICS/V5.

I dati appartenenti a destinazioni extrapartition consistono di records a lunghezza fissa o variabile, bloccati o sbloccati, il cui formato e' descritto nell'entrata di DCT che definisce ogni destinazione.

Al programmatore applicativo serve conoscere solo i quattro caratteri che costituiscono il nome simbolico della coda (DESTID) e, naturalmente, la lunghezza del record ed il relativo tracciato.

L'uso piu' comune delle destinazioni extrapartition e' quello di memorizzare dati per uso successivo fuori della regione CICS/VS; in particolare ricordiamo:

- il Data Entry (raccolta dati per successive elaborazioni batch);
- la creazione di un "giornale di fondo" delle transazioni (solitamente su nastro);
- la raccolta delle statistiche e dei messaggi di errore del CICS/VS.

Altro esempio: ove sia necessario stampare una grossa quantita' di dati, una destinazione extrapartition costituita da una stampatrice veloce in linea, puo' vantaggiosamente sostituire un terminale stampante, solitamente lento.

Poiche' nelle code i dati vengono scritti e letti sequenzialmente, e' possibile a piu' task memorizzare record nella stessa coda di output, ma, su una coda di input, e' bene che un solo task esegua operazioni di lettura.

10.4 DESTINAZIONI INTRAPARTITION

Le destinazioni intrapartition sono code di dati temporanei che sono memorizzati in un file DAM o VSAM ESDS e che vengono elaborate solo all'interno della regione CICS/VS. Cioe': sono create da un task ed elaborate da un altro task.

Il programmatore di sistema deve assegnare ad ogni destinazione un codice di identificazione (DESTID) di non piu' di quattro caratteri, ad elencarli in DCT.

Il file DAM o VSAM ESDS viene predisposto automaticamente, alla partenza del CICS/VS, in modo da riservare un certo spazio di disco ad ogni coda intrapartition elencata in DCT. Poi, secondo la disponibilita', il CICS/VS assegnera' automaticamente spazio alle code via via che ne avranno necessita'.

I record sono a lunghezza variabile (ma non superiori alla dimensione del Control Interval VSAM o della pista del device) e la memoria associata con una coda intrapartition puo' essere riutilizzata.

E' il programmatore di sistema che decide, per ogni destinazione, se lo spazio disco che essa occupa deve essere riusato non appena sono letti tutti i suoi records.

Se viene specificata come non riusabile, una coda intrapartition continua a crescere (indipendentemente dal fatto che i suoi dati siano già stati letti), finché non viene inviato un comando esplicito di cancellazione, che cancella l'intera coda e rilascia lo spazio su disco ad essa associato.

Alcuni degli impieghi più frequenti delle code intrapartition sono:

- Message Switching (invio di dati da un terminale ad un altro terminale);
- Diffusione di Messaggi (invio di messaggi a più terminali);
- Data Collection (raccolta di dati per elaborazione successiva).

Una destinazione intrapartition può essere usata da uno o più task contemporaneamente; è così possibile, per task diversi, anche se eseguiti nello stesso tempo, accodare dati sulla stessa destinazione. I dati sono elaborati dal TDP con tecnica FIFO (First-in, First-out), quindi il programma applicativo che scrive la coda non ha alcun controllo sulla sequenza finale dei records.

Inoltre, un record che è stato letto da una coda intrapartition, non può essere riletto da nessun task. Cioè, due task non possono leggere lo stesso record. Questo significa che, sebbene più task possano scrivere sulla stessa destinazione, solo un task dovrebbe leggere i dati di quella destinazione. Se, comunque, più task leggessero dalla stessa coda (considerando che essi non hanno il controllo della sequenza dei record) dovrebbero aspettarsi qualsiasi record, in accordo solo con la tecnica FIFO del TDP.

10.5 DESTINAZIONI INDIRETTE

Le destinazioni indirette sono dei riferimenti simbolici ad altre destinazioni fisiche, intrapartition o extrapartition.

L'uso di queste permette di accodare dati su una destinazione diversa da quella specificata nel programma, senza che si debba ricompilare il programma stesso. Sarà sufficiente modificare e ricatalogare la DCT.

Per realizzare questo, l'entrata in DCT relativa alla destinazione da modificare viene dichiarata "indiretta" ed è correlata (col parametro INDEST) ad un'altra entrata che rappresenta la nuova destinazione.

Quando la DCT è stata così modificata, il programma applicativo può inviare dati alla nuova destinazione, pur utilizzando il vecchio nome simbolico.

10.6 SCRITTURA SU TD

Poiche' le destinazioni indirette vengono realizzate per mezzo di entrate particolari nella DCT, solitamente al programmatore applicativo non interessa come questo venga fatto.

Per ulteriori informazioni vedi la "CICS/DS/V5 Resource Definition Guide".

10.6 SCRITTURA SU TD

Non vi e' differenza, dal punto di vista della programmazione, tra code extra ed intrapartition. Il comando di accodamento dei dati e':

```
EXEC CICS WRITEQ TD QUEUE (nome di coda)
    FROM (nome di campo)
    (LENGTH (numero))
```

QUEUE (nome di coda)
specifico il nome simbolico della destinazione come e' stato definito in DCT

FROM (nome di campo)
identifica l'area all'interno del programma in cui e' stato costruito il record

(LENGTH (numero))
indica la lunghezza del record da scrivere. Puo' essere omesso, per i record a lunghezza fissa da accodare su destinazioni extrapartition, se il programmatore conosce tale lunghezza e se l'area citata in FROM (nome di campo) e' stata definita con lunghezza identica a quella del record.

10.7 LETTURA DA TD

Anche per questo tipo di operazione non vi e' differenza tra code extra e intrapartition: il comando e' il medesimo, e fa sì che il primo record disponibile in testa alla coda venga reso accessibile al programma. La lettura e', ricordiamolo, sequenziale e distruttiva: il record letto viene tolto (logicamente e non fisicamente, ma agli effetti della programmazione le cose non cambiano) dalla coda, e quindi non puo' essere riletto.

```
EXEC CICS READQ TD QUEUE (nome di coda)
    (INTO (nome di campo))
    / SET (puntatore)
    (LENGTH (nome di campo))
```

QUEUE (nome di coda)
e' il nome simbolico della destinazione come definito in DCT.

INTO (nome di campo)

indica l'area all'interno del programma in cui deve essere letto il record.

SET (puntatore)

specifica il nome di un puntatore che deve essere impostato con l'indirizzo dei dati letti dalla coda. Vedi il cap. 14.

(LENGTH (nome di campo))

indica un campo di due byte binari in cui il programmatore imposta la lunghezza massima del record che si aspetta di ricevere. A lettura avvenuta, esso conterrà la lunghezza effettiva dei dati letti. Se la lunghezza specificata è inferiore a quella effettiva del record letto, questo viene troncato dando origine alla condizione di LENGTHR; nel campo viene impostata la lunghezza effettiva prima del troncamento. Il parametro LENGTH può essere omesso se si legge da destinazioni extrapartition aventi record di lunghezza fissa.

10.8 CANCELLAZIONE DI CODE DA TD

Abbiamo accennato al fatto che lo spazio disco occupato da una coda può esser riutilizzato non appena la coda sia stata letta. Per far ciò è necessario definire, nella DCT, la coda come "ri-usabile"; in caso contrario lo spazio su disco continuera' ad essere occupato anche se tutti i record della coda sono stati già letti e, di fatto, la coda non potrà più essere utilizzata (ricordiamo infatti che i record non possono essere riletti).

In realtà il meccanismo è un poco più complesso, ma non ci importa discuterne i dettagli: dobbiamo invece ricordare che, se una coda è stata definita nella DCT come non riuseabile, allora tocca a noi rilasciare lo spazio da essa occupato non appena avremo terminato di elaborarla.

Il comando da usare è'

EXEC CICS DELETEQ TD QUEUE (nome di coda)

Esso si applica solo alle destinazioni intrapartition, e cancella tutti i record della coda anche se non ancora letti.

10.9 PARTENZA AUTOMATICA DI TASK

Per le destinazioni intrapartition, il CICS/VS offre la possibilità di far partire automaticamente una transazione, usando un "trigger level" assegnato alla destinazione.

Un "trigger level" è semplicemente un valore numerico definito nella DCT dal programmatore di sistema e rappresenta il numero massimo di record che devono essere accumulati nella coda prima che parte la transazione collegata.

In pratica, quando il numero di record scritti nella coda (da comandi WRITEQ TD) raggiunge il "trigger level", il CICS/V5 automaticamente inizializza il task associato al codice di transazione specificato nella definizione della destinazione.

Questo nuovo task, associato o no ad un terminale, deve elaborare i dati della coda, inviando uno o più comandi READQ TD per svuotarla.

Una volta che la coda è stata svuotata, ricomincia il ciclo per una nuova partenza automatica. Per assicurarsi che la fine del task inizializzato automaticamente avvenga quando la coda è vuota, il programma applicativo dovrebbe verificare (tramite una HANDLE CONDITION) la condizione QZERO, piuttosto che usare qualche valore dipendente dall'applicazione.

10.10 CONDIZIONI ANOMALIE DI TD

Possiamo suddividerle in tre categorie.

1. Condizioni al verificarsi delle quali il programma "deve" prendere qualche decisione inserendo il comando HANDLE CONDITION.

Queste condizioni sono:

QZERO che si verifica quando una coda letta con il comando READQ TD è vuota

LENGERR che si verifica quando la lunghezza specificata è inferiore alla lunghezza effettiva dei dati letti.

2. Condizioni al verificarsi delle quali il programma "può" prendere qualche decisione.

HOSPACE non c'è più spazio disponibile su T.D. intrapartition per scrivere altri records

NTOOPEN si verifica se si cerca di accedere ad una destinazione extrapartition chiusa. Queste destinazioni possono essere aperte e chiuse dinamicamente dall'operatore di Master Terminal del CICS/V5.

QBUSY si verifica se si cerca di dare il comando READQ TD mentre un altro task sta scrivendo sulla coda. Tale condizione si manifesta solo per coda intrapartition. Il task che ha dato il comando READQ TD viene fatto attendere fino a che l'operazione di output in corso sulla coda non è terminata.

3. Condizioni all'insorgere delle quali è preferibile che il programma non prenda decisioni, ma lasci che il CICS/V5 chiuda abnormalmente il task.

IOERR errore di input/output

QIDERR il nome di coda specificato in una operazione di lettura o scrittura su TD non e' elencato in DCT.

10.11 SOMMARIO

Il Transient Data Program offre una funzione generalizzata per memorizzare dati temporanei in code sequenziali e, successivamente, per elaborarli.

"Code sequenziali" significa che i dati sono aggiunti in fondo alla coda e sono letti dall'inizio. I dati di una coda possono essere letti solo una volta.

Le code (o destinazioni) sono definite in Destination Control Table (DCT) dal programmatore di sistema, alla generazione del CICS/VS ed in accordo con le necessita' applicative.

I dati che sono creati ed elaborati da task CICS/VS vengono memorizzati in Destinazioni Intrapartition. I dati prodotti da programmi batch ed elaborati da task CICS/VS o, viceversa, i dati prodotti da task CICS/VS ed elaborati da programmi batch sono memorizzati in Destinazioni Extrapartition.

L'elaborazione di una coda puo' essere effettuata:

- in qualsiasi momento, usando un codice di transazione opportuno;
- automaticamente, (solo per code intrapartition) quando viene raggiunto un certo numero di record ("trigger level").

10. struttura e il suo funzionamento
11. struttura e il suo funzionamento

GERARCHIE DI CODICE

Per quanto riguarda questo tipo di struttura si intende che la struttura di programmazione sia così composta che il codice non debba essere scritto più di una volta per essere utilizzato in più di un luogo.

Questo significa che ogni struttura deve essere scritta una sola volta e deve essere utilizzata in più di un luogo.

Per esempio, se si ha un programma che deve essere eseguito in più di un luogo, questo programma deve essere scritto una sola volta e deve essere utilizzato in più di un luogo.

La struttura chiamata "funzione" può essere utilizzata per scrivere programmi che devono essere eseguiti in più di un luogo.

Il Temporary Storage rappresenta una struttura di memorizzazione degli dati temporanei che vengono creata e cancellata dinamicamente nel momento in cui vengono letti o scritti.

11.1 INTRODUZIONE

In questo capitolo vedremo le funzioni e i comandi del Temporary Storage del CICS/V5, che permettono ad un programma applicativo di gestire dei dati temporanei necessari, poi, allo stesso task o ad altri task.

Sotto questo aspetto e' simile al Transient Data, ma vedremo che il Temporary Storage offre ulteriori possibilita' rispetto a quelle viste nel capitolo precedente.

11.2 GENERALITA'

I dati che vogliamo gestire con il Temporary Storage sono scritti in forma di record a lunghezza variabile e memorizzati dal Temporary Storage Control Program (TSP) in memoria virtuale o in memoria ausiliaria (cioe' su disco), in funzione della scelta effettuata in fase di generazione.

In alcune installazioni sono disponibili sia la memoria ausiliaria che quella virtuale: il programmatore applicativo puo' usare il supporto che preferisce.

In genere la memoria virtuale e' bene utilizzarla per piccole quantita' di dati di utilizzo subitaneo, onde evitare problemi di disponibilita' di memoria e di perdita possibile di dati in caso di caduta del sistema.

La memoria ausiliaria e' costituita da un file VSAM aperto all'inizializzazione del CICS/V5 e disponibile per tutto il periodo relativo di attivita'.

Ad ogni record scritto su Temporary Storage deve essere associato un nome che serva ad identificarlo anche successivamente. Questo nome viene scelto dal programma applicativo nel momento stesso in cui scrive il record. L'unico vincolo e' che questo nome non sia piu' lungo di 8 caratteri.

Se piu' records verranno scritti usando lo stesso nome, questi formeranno una coda di Temporary Storage. I records di una coda possono essere letti sequenzialmente, nell'ordine in cui sono stati scritti, oppure direttamente, indicandone la posizione relativa all'interno della coda.

I nomi d'individuazione in TS dei record singoli o delle code devono essere univoci nell'intero sistema CICS/V5.

Poiche' piu' task potrebbero eseguire uno stesso programma, che scrive su TS, questo deve prevedere un sistema dinamico per creare nomi univoci per ogni task.

Un modo comune per far questo e' quello di includere il nome del terminale come parte del nome di TS. Ricordiamoci che il nome del terminale e' disponibile nel campo EIBTRMID del blocco EIB.

I dati memorizzati in TS sono disponibili per essere letti o modificati finche' non siano cancellati da una transazione utente. Percio' essi, a differenza di quelli su TD, possono essere esaminati piu' volte e da diversi task.

Le funzioni del TS, oltre che essere disponibili ad un programma applicativo, vengono utilizzate dal CICS/VIS per memorizzare:

- le pagine costruite dal Paging del BMS (vedi 11 Cap. 13);
- i dati per il controaggiornamento dinamico di transazioni terminate in modo anomalo (Dynamic Transaction Backout).

11.3 SCRITTURA SU TS

Per scrivere o modificare dati su di una coda di TS viene usato il seguente comando:

```
EXEC CICS WRITEQ TS QUEUE (nome di coda)
    FROM (nome di campo)
    LENGTH (numero)
    (ITEM (numero) (REWRITE))
    (MAIN/AUXILIARY)
```

QUEUE (nome di coda)

specificava il nome con cui vengono memorizzati i dati in coda. Può essere una costante o il nome di un'area lunga fino ad 8 caratteri, ed è arbitrario; il programmatore sceglie, per una coda, il nome che crede opportuno, con l'unico vincolo che esso deve essere univocamente determinato.

FROM (nome di campo)

indica l'area di programma in cui è stato costruito il record da scrivere su TS.

LENGTH (numero)

specificava la lunghezza del record da scrivere.

ITEM (numero) (REWRITE)

se nel comando si usa solo l'opzione ITEM, "numero" è un campo di due byte binari in cui il CICS/VIS imposta il numero d'ordine assegnato dal TS al record dopo averlo scritto. Se ITEM è usato insieme a REWRITE, "numero" è il valore numerico che individua la posizione all'interno della coda del record che si vuole aggiornare.

(il TS, a differenza del TD, ammette l'aggiornamento dei record).

MAIN / AUXILIARY

specifica se il record deve essere scritto in Memoria virtuale (MAIN) o su disco (AUXILIARY). L'assunzione implicita fatta dal CICS/V5 e' AUXILIARY.

E' conveniente usare la memoria virtuale se i dati devono essere memorizzati per un periodo breve. Per periodi lunghi la memoria auxiliaria e' piu' conveniente, anche perche' in caso di chiusura, normale o anomala, del CICS/V5, i dati non andranno perduti, e potranno essere ripresi alla successiva inizializzazione.

11.4 LETTURA DA TS

Per leggere i record di una data coda di TS viene usato il seguente comando:

```
EXEC CICS READQ TS QUEUE (nome di coda)
    (INTO (nome di campo)
     / SET (puntatore))
    LENGTH (nome di campo)
    (ITEM (numero)/NEXT)
    (NUMITEMS (nome di campo))
```

QUEUE (nome di coda)

specifica il nome simbolico della coda cui appartiene il record da leggere.

INTO (nome di campo)/SET (puntatore)

specifica il nome dell'area all'interno del programma in cui verrà letto il record. In alternativa, si può specificare un puntatore nell'opzione SET. In questo caso il CICS/V5 acquisisce un'area sufficiente per contenere il record ed imposta il puntatore all'indirizzo del record. (Vedi il Cap. 14).

LENGTH (nome di campo)

specifica un campo binario di due byte in cui occorre impostare la lunghezza massima del record che ci si aspetta di leggere. Dopo la lettura il campo conterrà la lunghezza effettiva del record letto. Se la lunghezza specificata dal programma è inferiore alla lunghezza del record, questo verrà troncato e si verificherà la condizione di LENGTH; il campo verrà impostato con la lunghezza effettiva prima del troncamento.

ITEM (numero) / NEXT

L'opzione NEXT viene usata per leggere sequenzialmente una coda. L'opzione ITEM, il cui argomento è il numero d'ordine del record che si desidera leggere, viene usata per accedere direttamente ai record della coda. In lettura sequenziale (opzione NEXT), la fine dell'

coda e' riconosciuta dal verificarsi delle condizioni ITEMERR. Il default e' NEXT.

HUMITEMS (nome di campo)

specifica un campo binario di due byte in cui il CICS/V5 memorizza un numero indicante quanti record ci siano nella coda.

L'opzione NEXT permette di leggere il record successivo all'ultimo letto, in assoluto, indipendentemente dal task. Percio', se piu' task devono accedere alla stessa coda ed ognuno deve cominciare la lettura dall'inizio, bisogna usare l'opzione ITEM.

11.5 CANCELLAZIONE DI CODE TS

Quando i record di una coda sono stati utilizzati, lo spazio da essi occupato puo' essere recuperato cancellandoli. L'opzione si esegue con il comando:

EXEC CICS DELETEQ TS QUEUE (nome di coda)

che cancella tutti i record della coda.

11.6 LE CONDIZIONI ANOMALIE DI TS

Possiamo suddividerle in tre categorie.

1. Condizioni al verificarsi delle quali il programma "deve" prendere qualche decisione inserendo il comando HANDLE CONDITION.

Queste condizioni sono:

ITEMERR che si verifica quando il numero di record specificato in un comando di READQ TS o WRITEQ TS e' invalido, o se non puo' essere localizzato il record da riscrivere specificato in un comando di WRITEQ TS REWRITE.

LENGERR che si verifica quando la lunghezza specificata e' inferiore alla lunghezza effettiva dei dati letti.

2. Condizioni al verificarsi delle quali il programma "puo'" prendere qualche decisione.

HOSPACE non c'e' piu' spazio disponibile sul data set TS per scrivere altri record. Se questa condizione non e' gestita il task viene sospeso fino a che non vi sia spazio disponibile.

3. Condizioni all'insorgere delle quali e' preferibile che il programma non prenda decisioni, ma lasci che il CICS/V5 chiuda abnormalmente il task.

IDERR errore irreparabile di I/O su TS
QIDERR il nome di coda specificato in una READQ TS o
 DELETEQ TS e' invalido
INVREQ i dati da scrivere su di una coda TS hanno lunghezza
 zero.

11.7 SOMMARIO

Il Temporary Storage e' l'altra possibilita' offerta dal CICS/VIS per memorizzare dati in una coda.

Le code sono solitamente su memoria ausiliaria, sebbene sia possibile scrivere code anche in memoria virtuale.

Un programma crea una coda dinamicamente, assegnandole un nome unico simbolico. In essa i record vengono scritti sequenzialmente, inviando opportuni comandi WRITEQ TS.

I record delle code di TS possono anche essere modificati (opzione REWRITE).

E' possibile fare letture (comando READQ) di tipo:

- sequenziali (opzione NEXT)
- dirette (opzione ITEM)

tante volte quanto si vuole. I record rimangono disponibili finche' non vengano cancellati esplicitamente (comando DELETEQ).

Le maggiori differenze tra code TS e code TD sono:

- nomi di code definiti dinamicamente;
- dati accessibili piu' volte;
- dati modificabili;
- dati accessibili anche direttamente;
- dati memorizzabili anche in memoria virtuale.

12.1 INTRODUZIONE

In questo capitolo vedremo alcune funzioni dell'Interval Control Program (ICP) e, in particolare, la possibilità di accedere all'orologio di sistema del CICS/V5 ed i comandi che consentono di gestire la partenza di task a tempi prestabiliti.

12.2 ORA E DATA DI UN TASK

Sappiamo che, per ogni task che nasce, viene creato un blocco EIB. Tra i campi in esso presenti, uno contiene l'ora in cui il task è iniziato (EIBTIME) ed un altro contiene la sua data (EIBDATE). Entrambi sono in formato decimale packed del tipo:

- 0HHMMSS+ per l'ora
- 0YYDDDD+ per la data

e possono essere letti dal programma applicativo in qualsiasi momento durante l'esecuzione del task.

Il CICS/V5 ci mette a disposizione un comando particolare che consente di aggiornare contemporaneamente i campi EIBTIME ed EIBDATE durante l'esecuzione del task. Il comando è:

EXEC CICS ASKTIME

Questo fa sì che i valori presenti in EIBTIME ed in EIBDATE vengano sostituiti dall'ora e, rispettivamente, dalla data corrente.

12.3 PARTENZA A TEMPO DI UN TASK

Finora conosciamo tre modi per iniziare un task:

- digitare un codice di transazione ad un terminale;
- specificare l'opzione TRANSID nel comando EXEC CICS RETURN;
- raggiungere un "trigger level" in una coda di TD intrapartition.

Il CICS/V5 ne fornisce un quarto: il comando START, che consente di far partire un task, collegato o meno ad un terminale, all'ora da noi specificata.

Il formato del comando è:

EXEC CICS START (INTERVAL (0)/INTERVAL (hhmmss)/
TIME (hhmmss))

TRANSID (nome)
(REQID (nome))
(FROM (nome di area) LENGTH (numero))
(TERMID (nome))
(RTRANSID (nome))
(CTERMID (nome))
(QUEUE (nome))

dove l'opzione obbligatoria TRANSID individua il codice della transazione che deve essere eseguita allo scadere del tempo indicato.

Inviando una serie di comandi START, ognuno indicante la stessa transazione e lo stesso terminale, e' possibile passare piu' record di dati al nuovo task che dovrà partire.

Il modo per indicare l'ora in cui si vuole far iniziare il task, puo' essere fornito in valore assoluto (opzione TIME) o in valore relativo (opzione INTERVAL).

Se non si codifica ne' TIME ne' INTERVAL, il task inizierà immediatamente dopo l'esecuzione del relativo comando di START.

L'opzione INTERVAL indica il periodo di tempo che deve trascorrere tra l'esecuzione del comando START e l'inizio del nuovo task. Il formato "hhmmss" indica:

- hh - numero di ore da zero a 99
- mm - numero di minuti da zero a 59
- ss - numero di secondi da zero a 59

L'opzione TIME indica, invece, l'ora del giorno in cui deve iniziare il task. Questo e' un tempo assoluto e potrebbe essere antecedente l'ora corrente.

Il sistema CICS/V5 usa un orologio suddiviso nelle solite 24 ore. Se un task deve partire, per esempio, alle tre del pomeriggio, TIME potrebbe essere (150000). Poiche' il CICS, pero', tratta come scaduto un tempo assoluto uguale all'ora corrente o che ne differisca meno di sei ore, se l'opzione TIME specificata precede l'ora corrente piu' di 6 ore, il CICS aggiunge 24 ore, cioè la funzione richiesta sera' elaborata all'ora stabilita ma il giorno dopo. Se, invece, l'opzione TIME cade entro le sei ore precedenti l'ora corrente, la funzione richiesta dal comando START sara' subito elaborata.

Nella prima pagina del Cap. 4.2. dell'APRM sono visibili alcuni esempi del comando START con l'opzione TIME.

Il task che lancia il comando START puo' indicare il terminale cui deve essere agganciato il nuovo task (opzione TERMINALE), oltre che la presenza di dati da passare (opzione FROM).

Altri dati possono essere passati al task invocato dal comando START facendo uso delle opzioni QUEUE, RTRANSID e RTERMID.

L'opzione QUEUE indica il nome di una coda che deve essere accessibile al nuovo task richiamato dal comando START.

Le opzioni RTRANSID e RTERMID indicano, rispettivamente, il codice di transazione e del terminale che vogliamo passare da un task ad un secondo task, richiamato dal comando START, perche' questo li utilizzi quando a sua volta richiamera' a tempo un terzo task.

L'opzione REQID specifica un nome che identifichi il comando. Essa serve, come vedremo (paragrafo 11.6), per poter cancellare una richiesta non ancora scaduta. Se quest'opzione viene omessa il CICS genera un identificatore univoco nel campo EIBREQID del blocco EIB.

12.4 LETTURA DEI DATI PASSATI AD UN TASK A TEMPO

I dati che devono essere trasferiti al task iniziato a tempo da un comando START sono memorizzati dal CICS/VIS su Temporary Storage ed e' possibile leggerli solo utilizzando il seguente comando:

```
EXEC CICS RETRIEVE (INTO (area-dati)/  
                      SET (puntatore))  
                      LENGTH (area-dati)  
                      (RTRANSID (area-dati))  
                      (RTERMID (area-dati))  
                      (QUEUE (area-dati))  
                      (WAIT)
```

L'opzione INTO e' usata per indicare l'area in cui devono essere messi i dati.

L'opzione LENGTH deve specificare un'area dati che contenga la lunghezza massima del record accettato dal programma applicativo. Se il record letto e' piu' lungo del massimo specificato, viene troncato e scatta la condizione di LENGTHERR. Dopo un comando RETRIEVE, l'area dati dell'opzione LENGTH contiene la lunghezza effettiva del record letto.

Alternativamente all'opzione INTO, si puo' specificare un puntatore nell'opzione SET. In questo caso il CICS/VIS acquisisce un'area sufficiente per contenere il record ed imposta il puntatore con l'indirizzo di tale area. (Vedi 11 Cap. 14).

Un task, che non sia collegato ad un terminale, puo' accedere solo al singolo record associato con il comando originale di START. Per questo e' necessario un solo comando RETRIEVE. L'area di T.S. occupata dal record associato con il task viene rilasciata dopo

I'esecuzione del comando RETRIEVE, oppure alla chiusura del task, se questo termina senza aver eseguito un comando RETRIEVE.

Un task, che sia collegato ad un terminale, puo' accedere a tutti i record associati con i relativi comandi START scaduti ed indicanti lo stesso codice di transazione e lo stesso terminale specificati nel comando START che ha innescato il task.

Per leggere questi record e' necessario inviare una serie di comandi RETRIEVE. A fronte di questi, i record sono presentati al task nell'ordine in cui sono scaduti, a partire da quelli memorizzati dal comando che ha innescato il task, ed includendo quelli provenienti dai comandi scaduti dopo l'inizio del task.

Quando tutti i record scaduti sono stati letti, scatta la condizione ENDDATA.

L'area di T.S. occupata da ogni singolo record associato con un comando START viene rilasciata dopo l'esecuzione di un comando RETRIEVE; tutta l'area di T.S. occupata da record scaduti, che non siano letti da comandi RETRIEVE, verrà rilasciata alla chiusura del CICS/V5.

L'opzione WAIT indica che, se tutti i record scaduti sono già stati letti, il task deve rimanere sospeso fino a quando divengano disponibili altri record scaduti.

Le opzioni QUEUE, RTERMID e RTRANSID devono indicare le aree dati in cui ricevere (durante l'esecuzione del comando RETRIEVE) i nomi relativi di Coda, Terminale e Transazione passati dal task che ha lanciato il comando START.

12.5 INTERRUZIONE DI UN TASK

Un task, durante la sua esecuzione, puo' richiedere al CICS/V5 di restare sospeso, per un certo periodo di tempo, e, allo scadere di questo, di riprendere l'elaborazione.

Il comando utilizzabile e':

```
EXEC CICS DELAY (INTERVAL (hhmmss)/  
                   INTERVAL (0)/  
                   TIME (hhmmss))  
                   (REQID (nome))
```

dove il periodo di sospensione puo' essere espresso in valore assoluto (opzione TIME) o in valore relativo (opzione INTERVAL). Se non si codifica ne' TIME ne' INTERVAL, il task sera' sospeso, ma richiederà subito al CICS di riprendere l'esecuzione.

L'opzione REQID puo' servire (in analogia con il comando START) per poter cancellare la richiesta di sospensione (vedi paragrafo 11.6).

12.6 CANCELLAZIONE DI RICHIESTE

Per poter cancellare una richiesta, precedentemente formulata tramite un comando START o DELAY, e' possibile utilizzare il seguente comando:

```
EXEC CICS CANCEL (REQID (nome)) (TRANSID (nome))
```

L'effetto della cancellazione varia, in funzione del tipo di comando da cancellare, nel seguente modo:

- Un comando START puo' essere cancellato, solo prima della sua scadenza, specificando l'opzione REQID del comando originale. In questo caso il comando CANCEL annulla la richiesta di partenza di un task effettuata con il comando START.
- Un comando DELAY puo' essere cancellato solo prima della sua scadenza e soltanto da un task diverso da quello che ha inviato il comando stesso di DELAY, ovviamente. Deve essere usato il medesimo REQID specificato dal task sospeso. L'effetto della cancellazione e' di far scadere immediatamente il tempo originale della sospensione; cioe' il task sospeso diventa subito pronto per essere attivato e ricominciare a lavorare.

12.7 LE CONDIZIONI ANOMALE DI IC

Possiamo dividerle in tre categorie.

1. Condizioni al verificarsi delle quali il programma DEVE prendere qualche decisione, inserendo il comando HANDLE CONDITION.

Queste condizioni sono:

ENDDATA si verifica se non ci sono piu' dati per il task che invia un comando RETRIEVE. Si puo' considerare un normale fine-file durante letture sequenziali di record.

ENVDEFERR si verifica ad un comando RETRIEVE che specifichi un'opzione non presente nel corrispondente comando START.

LENGERR si verifica ad un comando RETRIEVE INTO se la lunghezza specificata e' minore di quella effettiva dei dati memorizzati.

NOTFND si verifica in uno dei seguenti casi:

- * Il nome indicato nell'opzione REQID di un comando CANCEL e' sbagliato.

- Il comando RETRIEVE e' inviato da un task inizializzato da un comando START che non aveva specificato l'opzione FROM.
- Il nome specificato nell'opzione REQID di un comando START non e' univoco; quando viene inviato un comando RETRIEVE, i dati non possono essere trovati.

2. Condizioni al verificarsi delle quali il programma PUO' prendere qualche decisione:

EXPIRED si verifica se il tempo specificato in un comando DELAY e' gia' scaduto quando viene inviato il comando.

IDERR si verifica per un errore di input/output durante un comando RETRIEVE o START. L'operazione puo' essere ritentata con un nuovo comando RETRIEVE.

3. Condizioni al verificarsi delle quali e' preferibile che il programma lasci che il CICS/VS chiuda anormalmente il task:

INVREQ si verifica se il CICS/VS riceve un comando di "interval control" invalido.

INVTSREQ si verifica se, durante l'esecuzione di un comando RETRIEVE, non c'e' supporto per una lettura da Temporary Storage.

TERMIDERR si verifica se il terminale indicato in un comando START non esiste in TCT.

TRANSIDERR si verifica se il codice di transazione specificato in un comando START non esiste in PCT.

12.8 ALTRE POSSIBILITÀ

Finora abbiamo visto le funzioni piu' comuni offerte dall'Interval Control Program per la gestione dei task a tempo, vogliamo, pero', ricordare, per motivi di completezza, che esistono altri due comandi che possono essere utilizzati in situazioni particolari. Questi sono:

* il comando POST:

serve per chiedere al CICS/VS la disponibilita' di una "event control area", che venga opportunamente impostata quando il tempo specificato nel comando e' scaduto.

* il comando WAIT EVENT:

serve per sincronizzare un task con il completamento di un evento, che di solito consiste nell'impostare, allo scadere del tempo, la "event control area" richiesta da un comando POST.

Per una descrizione dettagliata di questi due comandi si rimanda al capitolo 4.2 dell'APRM.

t' bene ricordare anche che esistono delle funzioni, gestite dal Task Control Program, che risultano utili per controllare l'attività dei task e per gestire la condivisione di risorse particolari.

I comandi relativi sono:

SUSPEND serve per rilasciare il controllo ai task di piu' alta priorita' in attesa di essere eseguiti. Questo comando puo' essere usato per evitare che un task in esecuzione monopolizzi la CPU.

ENQ/DEQ servono per chiedere (o annullare) una prenotazione su una risorsa che debba essere protetta da possibili usi contemporanei da parte di piu' task.

L'uso di questi comandi e' necessario per tutte le risorse utente che non siano protette automaticamente dal CICS/V5; per esempio:

- tabelle di dati in memoria;
- campi di Common Work Area (CWA).

Per una descrizione dettagliata di questi tre comandi si rimanda al capitolo 4.3 dell'APRM.

12.9 SOMMARIO

All'inizio di questo capitolo abbiamo visto quali sono le possibilità di accedere all'ora di sistema e di aggiornare i campi EIBTIME e EIBDATE, contenenti l'ora e la data d'inizio del task in esecuzione (comando ASKTIME).

Abbiamo poi visto un quarto modo per far partire un task: l'uso del comando START. Con esso e' possibile far partire un task, collegato o meno ad un terminale, all'ora specificata nell'opzione TIME o dopo un certo intervallo di tempo (opzione INTERVAL). In un comando START e' possibile, inoltre, comunicare dei dati al nuovo task che dovrà partire, facendo uso delle opzioni FROM, QUEUE, RTRANSID e RTERMID. Questi dati potranno essere letti dal nuovo task tramite un comando RETRIEVE. Durante la sua esecuzione, un task puo' richiedere al CICS/V5 di restare sospeso

- per un certo periodo di tempo (comando DELAY);
- fino a quando ci siano task a priorita' maggiore in attesa di esecuzione (comando SUSPEND);
- fino al completamento di un evento (comando WAIT EVENT).

Facendo uso del comando CANCEL, e' sempre possibile cancellare una richiesta di partenza a tempo di un task (comando START) o di sospensione di un task (comando DELAY).

Alla fine del capitolo abbiamo accennato a due comandi particolare-
mente utili, chiamati ENQ e DEQ per gestire la condivisione di certe risorse utente
(per esempio tabelle in memoria), accessibili a piu' task contemporaneamente.

Lavori (Jobs) e Thread (Threads)

Nella parte di APLX la vengono ri-sintetizzate queste diverse entità:
dovevano essere chiamate entità di utente ed infatti
rispondono al nome di dati utente che sono prima "dati
di utente" e poi "dati di connessione".

Per esempio quando si creano dati di utente si creano anche dati di connessione
che sono associati ai dati di utente.

Si ottiene così una struttura "e connex" (lavoro) in cui
ogni struttura utente ha una struttura associata
che sono associate fra loro.

Le strutture di dati di connessione sono:

• COBOL work areas usate da coda.

Alcune di queste sono le stesse che vengono presentate nel
CAPITOLO 6. Eseguite le

Connessioni (Connections)

Queste si creano quando viene stabilita connessione di applicazione
tra il programma di applicazione di utente e quello che si creano
al posto dei dati di utente per ogni connessione (TAREA o WORKAREA o
WORKSTRUCTURE).

Queste sono definite con dati che devono essere dati specifici
come ad esempio dati relativi a dati di utente o dati di utente
o dati di connessione specifici come dimensioni delle varie strutture o dati
come ID dati, dati relativi alle istanze eseguite e TAREA numero
etc. Queste strutture sono definite attraverso "creare dati degli
utenti" (CREATE WORKAREAS) oppure attraverso TAREA o WORKAREA
di connessione con gli utenti (CREATE CONNECTIONS) oppure attraverso
CREATE CONNECTIONS.

Le strutture di connessione fanno parte di dati di utente e sono
CREATE CONNECTIONS attraverso i dati di utente.

Ciascuna di queste fanno parte di dati di utente.

13.1 INTRODUZIONE

In questo capitolo vedremo le cosiddette "funzioni avanzate del BMS" che forniscono al programmatore un validissimo supporto per costruire e gestire messaggi destinati ad uno o più utenti (o terminali).

13.2 MESSAGGI MULTIPAGINA

Nelle applicazioni di inquiry spesso si ricercano informazioni relative non ad un singolo elemento (persona, parte immagazzinata, movimento bancario, ecc.) ma a un gruppo più o meno numeroso di elementi; si ricercano, ad esempio, gli ultimi venti movimenti del conto corrente di un cliente di una banca, o le parti componenti di un certo articolo, o gli articoli, e relative quantità, di un certo ordine. La gestione di informazioni di tal tipo pone, dal punto di vista del formato con cui evidenziarle sui terminali, problemi nuovi rispetto a quelli che sono stati affrontati nei capitolo precedenti relativi al Terminal Control e al BMS.

Supponiamo, ad esempio, di disporre di un terminale video con schermo da 12 righe per 40 colonne, e di voler conoscere appunto gli ultimi venti movimenti di un certo conto corrente XYZ. Non riusciremo, con una singola operazione di output, ad evidenziare tutti i movimenti (nell'ipotesi che ciascuno di essi occupi almeno una riga dello schermo); saremo quindi costretti a spezzare il messaggio (costituito da una intestazione - comprendente numero di conto e intestatario - e dalle righe movimento), in due pagine.

Nel presentare questo breve esempio applicativo, abbiamo introdotto due termini (messaggio e pagina) che ci sembra opportuno definire subito con chiarezza; precisamente:

- intendiamo per messaggio - o meglio, per evitare confusioni con quanto già conosciamo - per messaggio logico, l'insieme delle informazioni, tra loro correlate, prodotte da una applicazione in risposta a richieste formulate da un operatore al terminale.
- per pagina intendiamo invece la frazione di messaggio logico che viene inviata con una singola operazione di output a terminale. Un messaggio logico è quindi costituito da una o più pagine la cui dimensione, in termine di "righe per colonne", viene specificata per ciascun terminale nella TCT.

La dimensione della pagina è ovviamente legata a quella del buffer del terminale; per i terminali video una pagina deve essere

contenuta entro il buffer del terminale stesso; per quelli stampanti la dimensione della riga della pagina non puo' superare quella fisica della riga di stampa.

13.3 STRUTTURA DELLE PAGINE

Di norma le pagine di un messaggio logico hanno tutte struttura simile, fatta eccezione, eventualmente, per la prima e per l'ultima. All'interno di ogni pagina e' possibile individuare elementi caratteristici quali:

- le informazioni di testata, presenti in ciascuna pagina
- le righe di dettaglio
- le informazioni di coda (o totali) presenti sicuramente nell'ultima pagina, ma che potrebbero comparire sotto forma di totali parziali anche in ogni altra pagina.

Notiamo inoltre (vedi figura 13.1) che le righe di dettaglio all'interno della pagina hanno tutte identico tracciato: sono, in altre parole, formalmente uguali.

Per la gestione dei messaggi multipagina e' conveniente riflettere la struttura della pagina nelle definizioni delle mappe BMS che vengono utilizzate per la preparazione del messaggio; durante la costruzione delle pagine si dovranno pertanto utilizzare mappe di testata, o HEADER, mappe di dettaglio e mappe di coda, o TRAILER. La circostanza si rivela in realtà molto pratica ed economica, in quanto consente di definire mappe relative a parti di pagina anziche' mappe relative alla pagina intera, e, com'e' noto, queste ultime sono piu' scomode da usare poiche' vincolate ad una certa dimensione di pagina.

13.4 USO DELLA DEVICE INDEPENDENCE

Una delle caratteristiche piu' rilevanti del BMS e' quella che consente al programmatore di svincolarsi, durante la preparazione di messaggi, dalle caratteristiche hardware dei terminali cui i dati sono indirizzati.

Tale particolarita' viene mantenuta, durante la costruzione di pagine, anche nei confronti della dimensione verticale della pagina (numero di righe) che viene definita in TCT e che quindi e' assimilabile ad una caratteristica del terminale.

Osserviamo infatti che, se si dispone di comandi cosi' flessibili da poter impartire al BMS istruzioni del tipo:

- posiziona le informazioni di testata, il cui tracciato e' descritto dalla mappa MAPPAL, all'inizio della pagina;

- posiziona la prima riga di dettaglio subito dopo la testata e le altre in sequenza, una dopo l'altra, ma riserva spazio in fondo alla pagina per le informazioni di coda. La mappa che descrive le righe di dettaglio si chiama MAPPA2;
- posiziona le informazioni di coda, la cui struttura è descritta dalla mappa MAPPA3, in fondo alla pagina;

siamo del tutto svincolati dal numero di righe costituenti la pagina.

Infine i termini:

- all'inizio
- in sequenza
- in fondo

hanno un significato flessibile, che si precise solo in riferimento a un ben definito terminale.

Tornando per un momento alla figura 12.1: "in fondo" significa alla riga 24, "dopo la testata" significa alla riga 7, e così via.

Tutto ciò è possibile usando la macro ed i comandi del BMS che già abbiamo visto ai capitoli 3 e 5, rispettivamente. Vedremo, comunque, che ci sono alcuni parametri addizionali ed un nuovo comando

13.5 POSIZIONAMENTO DI MAPPE

Riprendiamo la macro di definizione di una mappa (DFHMDI) già vista nel Cap. 3 ed esaminiamone i parametri omessi in quel capitolo.

Per quello che ci interessa in questo momento, il formato della macro DFHMDI è:

```
nome DFHMDI (COLUMN-(numero/NEXT/SAME),
              (LINE-(numero/NEXT/SAME),
               (SIZE-(linee.colonne),
                (JUSTIFY-((LEFT/RIGHT)),
                 ((FIRST/LAST)))),
               (HEADER=YES),
               (TRAILER=YES))
```

Tutti i parametri mancanti sono già stati esaminati nel Cap. 3; occupiamoci ora di quelli che si riferiscono al posizionamento delle mappe ed al loro uso.

In generale le mappe possono essere definite con una posizione assoluta sullo schermo (LINE=numero e COLUMN=numero) o con una posizione relativa ad altre mappe (LINE=SAME/NEXT e COLUMN=SAME/NEXT). Vediamo, quindi, tutti i parametri di posizionamento:

JUSTIFY=LEFT indica che i parametri di posizionamento sono relativi all'angolo superiore sinistro della mappa, e al margine verticale sinistro della pagina.

JUSTIFY=RIGHT indica che i parametri di posizionamento sono relativi all'angolo superiore destro della mappa, e al margine verticale destro della pagina.

JUSTIFY=FIRST indica che la mappa deve essere posizionata per prima su una nuova pagina. Il parametro puo' essere specificato sulle mappe di testata.

JUSTIFY=LAST indica che l'ultima riga della mappa deve essere posizionata nell'ultima riga della pagina. Il parametro puo' essere specificato sulle mappe di coda.

COLUMN=numero indica che l'angolo superiore (destro o sinistro) va allineato alla colonna indicate.

COLUMN=NEXT indica che la colonna di posizionamento deve essere quella successiva rispetto alla mappa precedente.

COLUMN=SAME indica che la colonna di posizionamento e' quella "corrente", cioe' l'ultima colonna specificata esplicitamente in una mappa posizionata in precedenza con **COLUMN=numero**.

LINE=numero indica la linea su cui posizionare l'angolo superiore (destro o sinistro) della mappa.

LINE=SAME indica che la linea di posizionamento e' quella corrente.

LINE=NEXT indica che la linea di posizionamento e' la prima linea disponibile completamente libera.

SIZE=(linee>colonne) indica la dimensione della mappa.

Naturalmente l'insieme dei tre parametri SIZE, COLUMN e LINE deve essere specificato in modo tale che la mappa possa essere contenuta nella pagina.

- I parametri LINE=... e JUSTIFY=LAST sono mutuamente esclusivi.
- L'uso di JUSTIFY=LAST svincola il Mapset dalle dimensioni effettive della pagina per un certo terminale.

Nel Cap. 3.2-4 dell'APRM ci sono diversi esempi di posizionamento delle mappe.

Una pagina puo' essere costituita di sole righe di dettaglio. Comunque, se vogliamo mettere dei dati in testa ed in fondo ad una pagina (vedi figura 12.1), dobbiamo definire almeno una mappa di testata ed una di coda.

HEADER=YES indica che la mappa e' una mappa di testata. Le mappe di testata sono sempre poste all'inizio di una pagina. Su una stessa pagina ce ne possono essere, una di seguito all'altra, piu' di una. Se, pero', una mappa di testata e' richiamata dopo una mappa non-di-testata, viene forzato l'inizio di una nuova pagina.

Notiamo che, mentre il parametro HEADER=YES va specificato per tutte le mappe di testata, JUSTIFY=FIRST va specificato solo sulla prima mappa di testata che si intende usare, poiche' esso forza l'acquisizione da parte del BMS di una nuova pagina (e' in sostanza il parametro che fa "volta pagina").

TRAILER=YES indica che la mappa e' una mappa di coda. Le mappe di coda possono essere posizionate in fondo alla pagina o subito dopo l'ultima mappa di dettaglio, in funzione dei parametri LINE o JUSTIFY. Piu' mappe di coda possono essere messe sulla stessa pagina.

Tutte le mappe (di testata, di dettaglio, di coda) che vengono usate per costruire una pagina di un messaggio logico possono appartenere a mapset diversi, ma e' piu' comodo se sono definite all'interno del medesimo MAPSET. Tra le mappe di coda, individuate dal parametro TRAILER=YES, che compaiono nel MAPSET, il BMS individua quella di dimensione verticale maggiore e riserva lo spazio di Trailer in base a tale dimensione e ai parametri LINE, COLUMN o JUSTIFY specificati per quella mappa.

Supponiamo che la mappa di TRAILER piu' grande nel MAPSET abbia:

SIZE=(2,20), LINE=23, COLUMN=1, TRAILER=YES

oppure:

SIZE=(2,20), JUSTIFY=LAST, COLUMN=1, TRAILER=YES

Lo spazio di Trailer e', in entrambi i casi (in una pagina di 24x80) di due righe: la 23a e la 24a.

Altro esempio:

SIZE=(2,20), LINE=20, COLUMN=1, TRAILER=YES.

In questo caso lo spazio di Trailer (sempre in una pagina 24x80) e' di cinque righe: dalla 20a alla 24a.

Nel caso che sia necessario utilizzare piu' mappe di coda nell'ambito della stessa pagina, possono nascere problemi sulla definizione dello spazio di Trailer.

Se le mappe di coda da definire insieme (di dimensioni uguali o diverse tra loro per numero di righe) sono allineate sulla stessa riga della pagina non ci sono problemi di definizione. Infatti, in questo caso lo spazio di Trailer, calcolato dal BMS per la mappa di coda piu' grande, potra' contenere anche le altre mappe di coda.

Nel caso in cui vi siano due mappe di coda posizionate su righe diverse (per esempio una alla riga 23 ed una alla riga 24 di una pagina 24 per 80), il BMS non riesce invece a definire uno spazio di TRAILER in modo univoco. Si deve ricorrere pertanto alla tecnica del TRAILER DUMMY, che si articola nelle seguenti fasi:

- si definisce nel mapset una mappa di Trailer, contenente almeno un campo, la cui dimensione sia tale da contenere i Trailer effettivi (nel nostro caso almeno 2 righe per il numero sufficiente di colonne).
- si specificano i parametri di posizionamento di questo Trailer Dummy in modo da riservare uno spazio di Trailer che coincida con le posizioni dei Trailer effettivi, (nell'esempio LINE=23 oppure JUSTIFY=LAST).
- Il Trailer Dummy non viene usato in nessun comando di BMS, ma viene solo utilizzato a livello di Mapset per il calcolo dello spazio di Trailer.

13.6 L'OVERFLOW DI PAGINA

Poiche' dunque il programma applicativo utilizza mappe la cui posizione all'interno della pagina non e' fissata esplicitamente, ne viene di conseguenza che esso e' indipendente dalla dimensione verticale della pagina (numero di righe).

Il programma e' in grado di costruire messaggi logici per tutti quei terminali che abbiano definito in TCT le righe delle rispettive pagine lunghe a sufficienza per contenere le righe del messaggio, indipendentemente dal numero di righe che la pagina di ciascun terminale contiene.

Ma se il programma applicativo ignora la dimensione della pagina che sta costruendo, come e' possibile collocare correttamente le informazioni di coda?

Sappiamo che la dimensione della pagina e' definita nella TCT, e che la mappa di coda viene costruita usando una macroistruzione in cui compaiono come parametri il numero di righe e di colonne occu-

Pate; orbene sono queste due informazioni che consentono al BMS di notificare al programma applicativo il momento di completare la Pagina con le informazioni di coda.

Il programma deve codificare una opportuna HANDLE CONDITION indicando la routine che deve assumere il controllo quando si verifica la condizione di "pagina piena" (OVERFLOW).

L'OVERFLOW si verifica quando il programma tenta di posizionare mappe entro lo spazio riservato alle informazioni di coda.

Nel paragrafo 12.8 vedremo in dettaglio cosa deve fare la routine di "overflow". Possiamo, pero', gia' dire che essa deve lanciare il comando di posizionamento della mappa di coda (per completare la pagina) e della testata per la nuova pagina e, poi, restituire il controllo al comando di scrittura della mappa di dettaglio che ha provocato l'"overflow".

Ancora si noti che, al momento di cedere il controllo alla routine di "overflow", il BMS e' in grado di fornire il numero della pagina in via di completamento.

13.7 COSTRUZIONE DI MESSAGGI LOGICI

Una pagina di output, costituita da piu' mappe, viene realizzata inviando dal programma applicativo, una sequenza opportuna di comandi SEND MAP, ognuno indicante la mappa necessaria per riempire una particolare porzione della pagina.

La sequenza dei comandi di output deve essere tale da riempire la pagina dall'alto in basso e da sinistra a destra.

Nel caso piu' semplice il messaggio logico da costruire e' formato da una sola pagina. Allora il comando SEND MAP, per quello che ci riguarda, e':

```
EXEC CICS SEND MAP (nome)
      (MAPSET (nome))
      TERMINAL
      ACCUM
      ERASE
```

dove tutti i parametri omessi sono quelli già visti al Cap. 5; qui vogliamo evidenziare la presenza della nuova opzione ACCUM.

Essa indica che la mappa specificata nella opzione MAP deve essere usata, insieme con altre mappe, per costruire una pagina.

Se si costruisce un messaggio logico in cui ogni pagina e' costituita da una sola mappa, la codifica dell'opzione ACCUM non e' piu' necessaria (caso già visto nel Cap. 5).

La pagina completata deve essere trasmessa al terminale: l'opzione TERMINAL e' anche il default.

Ricordiamo che l'opzione ERASE provoca l'azzeramento del buffer del terminale, prima che la pagina venga visualizzata, e dovrebbe essere sempre specificata nella prima operazione di output. Se l'opzione ERASE viene incontrata piu' volte nei comandi di costruzione di una pagina, sara' sempre ignorata, tranne la prima volta.

Nel caso piu' generale, pero', un messaggio logico e' costituito di piu' pagine.

Allora, per realizzare un messaggio multi-pagina, utilizzeremo dei comandi di output del tipo:

```
EXEC CICS SEND MAP (nome)
      (MAPSET (nome))
      (TERMINAL/PAGING/SET (puntatore))
      (ACCUM)
      (REQID (nome))
```

dove tutti i parametri omessi sono già stati illustrati nel Cap. 5, relativo a mappe singole.

Le opzioni nuove, oltre ad ACCUM appena vista, sono: PAGING/SET (puntatore) e REQID (nome).

Le prime due determinano, in alternativa tra di loro e con TERMINAL, la destinazione del messaggio logico.

PAGING indica che le pagine costituenti il messaggio logico devono essere memorizzate in Temporary Storage, e verranno successivamente esaminate dall'operatore al terminale. Deve essere specificato in tutti i comandi che creano lo stesso messaggio logico su TS.

SET(puntatore) specifica un puntatore (vedi il cap.14) che deve essere impostato con l'indirizzo di una lista di pagine di output completate (dipendenti dal tipo di terminale), che sono messe a disposizione del programma applicativo. Alla fine del Cap. 3.2-4 dell'APRM e' possibile reperire informazioni dettagliate sull'opzione SET e sul suo utilizzo.

TERMINAL

come abbiamo gia' detto, viene assunto implicitamente se nel comando SEND MAP non viene codificato ne' PAGING ne' SET, ed indica che le pagine del messaggio logico devono essere inviate immediatamente al terminale, via via che sono complete. L'operatore, quindi, non ha la possibilità di scorrerle a suo piacere.

Per i terminali video l'opzione da codificare e' PAGING, per poter memorizzare le pagine in Temporary Storage e consentire, successivamente, all'operatore di scorrerle avanti e indietro (vedi il paragrafo successivo 13.11). Mentre per i terminali stampanti l'opzione normale e' TERMINAL, poiche' il supporto cartaceo consente di esaminare le pagine in ogni momento.

ACCUM

come gia' detto, serve nel caso che piu' mappe debbano riempire una stessa pagina. Se non e' specificata, ogni comando di output creera' una pagina completa.

REQID (nome)

specifica un prefisso di due byte usato come parte di un identificatore di TS. Esso serve per garantire la ricoverabilita' di un messaggio logico creato su TS con l'uso dell'opzione PAGING. Il prefisso di default e' "XX". Su questo argomento puoi leggere un paragrafo apposito all'inizio del capitolo 3.2-4 dell'APRM.

13.8 GESTIONE DELL'OVERFLOW

Come abbiamo detto al paragrafo 12.5, durante la costruzione di un messaggio logico, il BMS riserva fin dal primo comando SEND MAP, uno spazio di "trailer", sufficiente per contenere una qualsiasi mappa di coda che possa essere inclusa nella pagina in costruzione.

Lo spazio rimanente e' disponibile per accomodare le mappe di dettaglio. Man mano che i dati vengono accumulati nel buffer della pagina, in risposta alla sequenza di comandi SEND MAP, il BMS tiene traccia dello spazio rimanente. Quando non c'e' piu' spazio disponibile per contenere una nuova mappa di dettaglio, il successivo comando SEND MAP fa scattare la condizione di "overflow" di pagina.

Per gestire questa condizione il programma deve codificare:

EXEC CICS HANDLE CONDITION OVERFLOW (label)

prima di iniziare la costruzione del messaggio logico.

Non appena si verifica la condizione di "overflow", il controllo viene assunto dalla routine la cui label di inizio compare come argomento di OVERFLOW. Essa puo':

- conoscere il numero di pagine costruite fino a quel momento mediante il comando:

EXEC CICS ASSIGN PAGENUM (campo)

dove "campo" e' definito come una mezza voce binaria: PIC S9(4) COMP..

- posizionare la o le eventuali mappe di coda nello spazio di TRAILER (se ne e' stato riservato uno) con un comando SEND MAP ACCUM
- posizionare la o le eventuali mappe di testata sulla nuova pagina con un comando SEND MAP ACCUM ERASE.
- ritornare al programma applicativo in modo da scrivere la mappa di dettaglio che ha causato l'overflow (e che verrà posizionata sulla nuova pagina).

Se l'overflow di pagina non viene gestito, la mappa di dettaglio che ha provocato la condizione di overflow verrà scritta automaticamente su di una nuova pagina; e' da notare che in tal caso la vecchia pagina non contiene informazioni di coda e la nuova non contiene informazioni di testata.

Tutti i dati impostati nella mappa che ha causato l'overflow restano inalterati e disponibili per tutto il tempo che il programma invia comandi BMS relativi solamente a mappe di testata o di coda.

13.9 CHIUSURA DI UN MESSAGGIO LOGICO

La fine di un messaggio logico deve essere comunicato al BMS con un comando apposito, il cui formato e':

```
EXEC CICS SEND PAGE (AUTOPAGE/NOAUTOPAGE)
                      (OPERPURGE)
                      (RELEASE (TRANSID (nome))/
                      RETAIN)
```

Questo comando chiude il messaggio logico in corso ed innesca la transazione di Paginazione, che forza il terminale nello "stato di paginazione", permettendo, così, la visualizzazione delle pagine del messaggio (vedi più avanti il paragrafo 12.11).

Le varie opzioni servono per specificare:

- quanto controllo dovrebbe avere l'operatore sulla disposizione del messaggio logico (AUTOPAGE, NOAUTOPAGE e OPERPURGE);
- se il controllo debba tornare al programma applicativo dopo la trasmissione del messaggio logico (opzione RETAIN) oppure no (opzione RELEASE).

AUTOPAGE

specificata che ogni pagina del messaggio logico deve essere inviata al terminale, non appena esso sia disponibile, senza intervento da parte dell'operatore. AUTOPAGE e' assunto per stampanti 3270; mentre non si applica a terminali video 3270.

NOAUTOPAGE

richiede che le pagine vengano inviate al terminale, una alla volta, su esplicita richiesta dell'operatore, che utilizzerà a tal fine gli appropriati Comandi di Paginazione (vedi il paragrafo 12.11). La prima pagina del messaggio viene inviata automaticamente. NOAUTOPAGE e' usato normalmente per i terminali video; mentre non va utilizzato per le stampanti 3270.

- Se ne' AUTOPAGE ne' NOAUTOPAGE vengono specificate, il messaggio logico verrà rilasciato in accordo con lo stato di paginazione specificato per il terminale nella relativa entrata di TCT.

OPERPURGE

indica che le pagine del messaggio logico devono essere cancellate dalla coda di Temporary Storage con un esplicito comando immesso dall'operatore.

- Alla esecuzione della SEND PAGE, il terminale cui e' agganciato il task che emette tale comando viene posto dal CICS/V5 in stato di paginazione; da quell'istante in poi il CICS/V5 accetta dal terminale solamente i comandi di paginazione (vedi il paragrafo 12.11). Dal terminale si potranno immettere codici di transazione ed altri dati solo quando non vi saranno per esso piu' pagine in coda. Se e' stato codificato OPERPURGE, le pagine possono essere cancellate con un comando appropriato, che come effetto secondario ha quello di far uscire il terminale dallo stato di paginazione. Se OPERPURGE non e' stato codificato, non appena l'operatore immette un messaggio non riconosciuto dal CICS/V5 come comando di paginazione, il terminale esce dallo stato di paginazione e le pagine in coda vengono cancellate.
- Le opzioni RETAIN e RELEASE hanno significato solo se si riferiscono a messaggi logici costruiti su Temporary Storage (opzione PAGING).

RETAIN

se e' stata codificata questa opzione, il programma riprenderà il controllo all'istruzione successiva alla SEND PAGE, non appena il messaggio logico in coda per il terminale sarà stato letto.

RELEASE (TRANSID (nome)) richiede che il programma, dopo l'esecuzione completa del comando SEND PAGE, restituisca il controllo al livello logico

superiore, subito dopo che le pagine sono state scritte al terminale. Se il programma e' al livello logico piu' alto, la sua fine coincide con la fine del task, ed in tal caso e' possibile specificare il codice della transazione che verrà agganciata al terminale all'"interrupt" successivo, in modo del tutto analogo a quanto si puo' fare con il comando di Program Control EXEC CICS RETURN TRANSID (nome).

- Se non si codifica né RETAIN né RELEASE, il controllo, dopo l'esecuzione della SEND PAGE, ritorna al programma applicativo e le pagine verranno inviate a terminali alla fine del task; cioè la Transazione di Paginazione resterà sospesa fino alla fine del programma in esecuzione.

13.10 LA TRANSAZIONE DI PAGINAZIONE

Con il comando SEND PAGE, la cui sintassi abbiamo appena vista al paragrafo precedente, il programma applicativo termina il messaggio logico ed inizia la Transazione di Paginazione.

Se per questo messaggio logico abbiamo usato l'opzione PAGING, il comando SEND PAGE scrive l'ultima pagina su TS e, quindi, inizia la Transazione di Paginazione, controllata dalle opzioni RETAIN e RELEASE.

- Se e' specificato RETAIN, la transazione di paginazione comincia subito. Viene inviata al terminale la prima pagina e l'operatore puo' usare i comandi di paginazione (vedi più avanti). In questo caso la transazione di paginazione opera in modo conversazionale. Quando l'operatore termina la sessione di paginazione, cancellando il messaggio, il controllo torna al programma al comando successivo al SEND PAGE RETAIN.
- Quindi la sessione di paginazione termina solo con la cancellazione del messaggio logico. Come già detto, con l'opzione OPERPURGE del comando SEND PAGE, il CICS cancellera' il messaggio logico solo su esplicito comando dell'operatore; mentre se l'opzione OPERPURGE e' omessa, il messaggio logico verrà cancellato appena l'operatore immette dei dati non riconosciuti come comandi di paginazione. Questi dati potrebbero essere letti dal programma applicativo, se, di seguito al comando SEND PAGE RETAIN, e' codificato un comando RECEIVE MAP.
- Se e' specificato RELEASE, la transazione di paginazione visualizza subito la prima pagina dei dati. Quindi la transazione del programma applicativo termina e la sessione di paginazione continua in modo pseudo-conversazionale. Alla fine della sessione di paginazione (con la cancellazione del messaggio logico), la transazione di paginazione restituisce

il controllo al CICS/VS che potra' iniziare, eventualmente, la nuova transazione specificata, nell'opzione TRANSID del comando SEND PAGE RELEASE, in chiusura del programma applicativo. Chiaramente il comando SEND PAGE RELEASE deve essere l'ultimo della transazione. Esso puo' essere pensato come una combinazione dei comandi SEND PAGE RETAIN e RETURN.

- Se non sono specificati ne' RETAIN ne' RELEASE, la transazione di paginazione viene iniziata quando termina la transazione applicativa corrente. Nel frattempo la transazione di paginazione viene accodata sul terminale destinatario, di seguito a tutte le altre transazioni gia' accodate per lo stesso terminale.

13.11 COMANDI DI PAGINAZIONE

I comandi di paginazione forniti dalla transazione di paginazione sono interamente discussi e illustrati nella dispensa "CICS/OS/VS Operator's Guide". Qui accenniamo solo alle funzioni messe a disposizione dell'operatore al terminale per gestire i messaggi logici.

Esse sono:

- Leggere un messaggio: L'operatore puo' digitare un comando (oppure usare un tasto PF opportuno) per leggere la prima, l'ultima, l'ennesima, la prossima o la precedente pagina di un messaggio logico.
- Chiedere la lista dei messaggi: Con un comando apposito l'operatore puo' ottenere la lista dei messaggi logici accodati per il suo terminale. Questa lista include l'identificatore del messaggio, che e' possibile utilizzare per leggere il contenuto del messaggio stesso.
- Copiare un messaggio: L'operatore puo' copiare la pagina corrente di un messaggio logico su un altro terminale (generalmente una stampante).
- Concatenare un messaggio: Durante una sessione di paginazione, l'operatore puo' invocare un'altra transazione, che comunichera' con il terminale nel modo usuale. Questa transazione invocata puo', a sua volta, costruire pagine, che saranno "concatenate" alle pagine costruite dalla transazione iniziale (se la transazione invocata ha specificato RETAIN o RELEASE nel suo comando SEND PAGE).
- Cancellare un messaggio: L'operatore indica di aver terminato la lettura di un messaggio logico cancellandolo. Questo, come gia' detto nei due paragrafi precedenti, puo' essere fatto esplicitamente, con l'uso di un comando di cancellazione, oppure implicitamente, immettendo dati che non siano comandi di paginazione, in funzione della presenza o meno

della opzione OPERPURGE nel comando SEND PAGE che ha iniziato la sessione di paginazione.

13.12 CANCELLAZIONE DI UN MESSAGGIO LOGICO

Se il programma stabilisce che le pagine di un messaggio, non ancora chiuso dalla SEND PAGE, debbano essere per qualunque motivo cancellate da Temporary Storage, deve codificare il comando:

EXEC CICS PURGE MESSAGE

Ricordiamo che le pagine vengono memorizzate su Temporary Storage solo se e' stato codificato PAGING nelle SEND MAP usate per la loro costruzione.

13.13 COSTRUZIONE DI TESTI

La possibilita' di "costruire testi", cioè di approntare messaggi senza un formato predefinito, risulta utile in alcuni casi in cui e' impossibile conoscere a priori la struttura dei dati, la loro natura e la loro dimensione.

Si considerino ad esempio le transazioni note come "Message Switching", consistenti nella formulazione manuale di un messaggio che l'operatore al terminale A vuole inviare all'operatore al terminale B, o che l'operatore con il controllo operativo di tutta la rete vuole inviare a tutti, o a un particolare gruppo di operatori.

La natura del messaggio e la sua forma variano di caso in caso, secondo le esigenze operative.

Quando non e' possibile conoscere il tracciato di un messaggio, non e' nemmeno possibile, evidentemente, associare ad esso un Mapset contenente le mappe che ne descrivono le parti. In questo caso e' ancora possibile usufruire dei vantaggi della "device independence" del BMS, usando il comando SEND TEXT, che permette di "formattare" i dati di output senza fare uso di mappe.

Il programma applicativo invoca l'ausilio delle funzioni di "text" del BMS eseguendo un comando SEND TEXT e specificando un'area "dati di output in cui sono già memorizzati i dati da inviare.

La stringa dei dati non avendo formato, che vogliamo inviare ad un terminale (video o stampante), viene divisa dal BMS in righe di lunghezza uguale a quanto definito nella TCT per il terminale ricevente.

I caratteri "blank" presenti nella stringa dei dati sono usati per separare le parole: se una parola intera non entra nello spazio ancora disponibile di una riga, viene messa interamente nella riga successiva.

Il BMS, inoltre, inserisce un byte attributo all'inizio di ogni riga, definendola come un campo non protetto e di normale intensità luminosa.

Questo fa sì che ogni riga di testo, visualizzabile o stampabile, sia un carattere più corto della lunghezza fisica di riga definita per il terminale.

Con terminali che supportano gli attributi dei campi, è possibile fare uso a programma di particolari ordini di "set attribute" (SA) per gestire dinamicamente gli attributi durante la costruzione di testi (vedi l'inizio del Cap. 3.2-3 dell'APRM).

I dati che devono essere "formattati" possono essere presentati al BMS con una singola richiesta (comando TEXT isolato) oppure con più richieste (sequenza di comandi TEXT).

Nel primo caso i dati vengono trasmessi al terminale appena l'operazione di "formattazione" della singola richiesta di output è completa.

Nel secondo caso le richieste possono essere accumulate per formare una o più pagine di output, in funzione della lunghezza dei dati stessi.

In questo caso è possibile specificare anche informazioni di testata e di coda, che siano però già definite nel programma (vedi più avanti il formato del comando SEND TEXT).

Un'altra caratteristica, nell'uso della funzione di "text" del BMS, è la possibilità di includere nel programma applicativo i caratteri "new-line" (NL) e "end-of-message" (EOM) nella stringa dei dati da "formattare", per gestire la spaziatura di righe brevi, paragrafi ed altri dati irregolari.

Poiché il BMS riconosce, come l'inizio di una stringa di dati da "formattare", il primo carattere diverso da "blank", se vogliamo che all'inizio di una riga compaiano dei "blank", è necessario codificare un carattere non-blank seguito dal numero desiderato di "blank".

Per quanto scopo possiamo utilizzare il carattere NL, riconosciuto dal BMS come X'15', ed inserirli ovunque nella stringa dei dati da formattare.

Ogni volta che il BMS incontra un carattere 'NL, viene chiusa la riga corrente ed iniziata una nuova. Una sequenza di NL provocherà, in output, una serie di righe bianche.

Per dati destinati a stampanti 3270, è possibile fare uso di due valori simbolici presenti nella lista DFHBMSCA, copiata nel programma:

- DFHBMPEM per EOM

• DFHBMPNL per NL

Riguarda il paragrafo 5.4.3 del Cap. 5.

Di solito il corpo del testo che deve essere "formattato" viene costruito in un'area di lavoro del programma. Quest'area deve essere sufficientemente ampia per contenere i dati utente ed ogni carattere di controllo desiderato. Sarà poi il BMS che muoverà i dati da formattare in un buffer di dimensione della pagina del terminale e creerà le pagine di output necessarie per contenere il testo finale.

Immaginiamo di voler realizzare il testo visibile in fig. 13.2.

Nel caso più semplice in cui il testo di figura 13.2 non sia modificabile, è possibile definire a programma un'area di lavoro, impostata con una serie di costanti, del tipo:

```
01 TEXTDATA.  
02 NL1    PIC X.  
02 FILLER PIC X(18) VALUE 'QUESTO TESTO SERVE'.  
02 NL2    PIC X.  
02 NL3    PIC X.  
02 FILLER PIC X(21) VALUE ' COME ESEMPIO DI USO'.  
02 NL4    PIC X.  
02 NL5    PIC X.  
02 FILLER PIC X(20) VALUE ' DEL COMANDO TEXT'.  
02 NL6    PIC X.
```

13.14 IL COMANDO SEND TEXT

Il formato del comando che invia un testo senza l'uso di mappe è:

```
EXEC CICS SEND TEXT FROM (campo)  
      LENGTH (numero)  
      (PAGING/SET (puntatore)/TERMINAL)  
      (ACCUM)  
      (HEADER (campo))  
      (JUSTIFY (numero)/JUSFIRST/JUSLAST)  
      (REQID (nome))  
      (TRAILER (campo))  
      (ALARM)  
      (CURSOR (numero))  
      (ERASE)  
      (FREEKB)  
      (HOMEOM/L40/L64/L80)  
      (NLEOM)  
      (PRINT)  
      (FORMFEED)
```

Rispetto al comando di SEND MAP già noto (vedi il Cap. 5 e il paragrafo 12.7) compaiono dei parametri nuovi, ai quali limiteremo la illustrazione che segue, in quanto gli altri conservano

in questo comando SEND TEXT il significato già visto in precedenza.

FROM (campo) indica l'area di programma che contiene i dati da trasmettere. La suddivisione in pagine viene fatta in base alla dimensione della pagina definita in TCT.

LENGTH (numero) indica la lunghezza dei dati del messaggio.

Nell'esempio del paragrafo precedente, il comando per inviare il testo al terminale è semplicemente:

```
EXEC CICS SEND TEXT FROM (TEXTDATA)
      LENGTH (65)
      ERASE
```

Se, invece, abbiamo più gruppi di testi che vogliamo accumulare in una pagina di output, faremo uso anche dell'opzione ACCUM (già nota) ed eventualmente, per messaggi logici da scrivere su TS, anche dell'opzione PAGING.

La condizione di "overflow" non è sentita da un comando SEND TEXT e non c'è un modo semplice per gestire a programma la fine di ogni pagina di output. Un modo potrebbe essere quello di usare l'opzione SET e la HANDLE CONDITION RETPAGE, come è discusso in un apposito paragrafo del Cap. 3.2-4 dell'APRM.

E' possibile però chiedere al BMS che certe informazioni fisse di testata e/o di coda siano inserite automaticamente all'inizio e/o alla fine di ogni pagina di output. Per questo sono disponibili le opzioni HEADER e TRAILER, che devono essere ripetute in ogni comando SEND TEXT ACCUM del messaggio logico.

Il BMS, inoltre, numera automaticamente le pagine di output, se gli viene definita opportunamente la testata o la coda.

Le righe di testata e di coda devono essere definite come aree distinte nella Working Storage Section del programma ed indicate come argomenti delle opzioni HEADER e TRAILER. Entrambe le aree hanno il seguente formato:

```
-----+-----+-----+-----+-----+
 | LL | # | R |   DATI   | ##### | DATI  |
 -----+-----+-----+-----+-----+
    2   1   1 |<---- Dati utente ----->|
```

in cui:

LL è un campo di due byte che contiene in binario la lunghezza dei dati utente più i due byte di questo campo.

T e' un campo di un byte che contiene un carattere identificatore di campo. Esso puo' essere un qualunque carattere eccetto: X'0C', X'15', X'17', X'26', X'FF'. Tale carattere consente al BMS di identificare, all'interno dei dati della testata o della coda, una sequenza da uno a cinque caratteri uguali all'identificatore, che definiscono un campo in cui le pagine vengono automaticamente numerate. Se non si vuole la numerazione, il designatore deve essere X'40' (blank).

R e' il byte riservato.

Se si desiderano testate su piu' di una riga, e' possibile inserire il carattere di New Line (X'15') all'interno dei dati.

8 8 8 8 questo e' il campo in cui il BMS mette automaticamente il numero di pagina (max 5 posizioni)

JUSTIFY (numero) indica la riga della pagina su cui iniziare a posizionare i dati. Se esiste una testata, la prima riga dati segue la testata. "numero" puo' assumere valori compresi tra 1 e 240.

JUSFIRST indica che i dati debbono iniziare dalla prima linea della pagina, o dalla prima linea dopo la testata, se questa esiste.

JUSLAST indica che i dati debbono essere posizionati in fondo alla pagina, o immediatamente prima del Trailer, se questo esiste.

13.15 CHIUSURA DI UN TESTO

Un messaggio logico costruito con comandi SEND TEXT ACCUM deve essere chiuso con un comando SEND PAGE, come gia' sappiamo.

Il comando SEND PAGE ha ora il formato:

```
EXEC CICS SEND PAGE (AUTOPAGE/NOAUTOPAGE)
                      (OPERPURGE)
                      (RELEASE (TRANSID (name))/RETAIN)
                      (TRAILER (campo))
```

e, come si vede, consente di posizionare sull'ultima pagina un TRAILER.

Infatti, poiche', la coda specificata nei comandi SEND TEXT viene inserita solo su tutte le pagine complete del messaggio logico, l'uso dell'opzione TRAILER nel comando SEND PAGE permette di inserire una coda anche nell'ultima pagina parzialmente completa.

E' possibile inoltre utilizzare il comando:

EXEC CICS PURGE MESSAGE

Per cancellare un messaggio parzialmente costruito, di cui si riscontra l'inutilità.

Informazioni esaurienti sull'uso del comando SEND TEXT sono reperibili nel Cap. 3.2-4 dell'APRM.

13.16 IL ROUTING

Col termine "Routing" si intende la costruzione di un messaggio logico da inviare a uno o più terminali diversi da quello cui è agganciato il task che produce il messaggio logico. Come caso particolare, tra i terminali destinatari del messaggio può esservi anche quello cui è legato il task; esso viene talvolta indicato come "terminale diretto" (Direct Terminal) mentre gli altri vengono indicati come "terminali destinatari" (Routed Terminal).

Durante un Routing, la costruzione delle pagine di messaggio non differisce da quanto visto nei precedenti paragrafi: si utilizzano i comandi SEND MAP, SEND TEXT, SEND PAGE seguendo le regole illustrate, e tenendo presente che la disposizione da codificare è PAGING, salvo esigenze particolari in cui si sceglie SET. Non può invece essere usata la disposizione TERMINAL, poiché essa ha per effetto di sottrarre al Routing i dati coinvolti nel comando relativo e di inviarli al terminale diretto.

Caratteristico del Routing è il comando ROUTE con cui si comunica al BMS la lista dei terminali e operatori destinatari e la modalità di invio del messaggio.

Esso va codificato prima di iniziare la costruzione del messaggio stesso, ed ha il seguente formato:

```
EXEC CICS ROUTE (INTERVAL (hhmmss)/INTERVAL(0)
                  /TIME (hhmmss))
                  (ERRTERM (nome))
                  (OPCLASS (campo))
                  (LIST (campo))
                  (TITLE (campo))
```

INTERVAL (hhmmss) indica l'intervallo di tempo dopo cui inviare il messaggio

TIME (hhmmss) indica l'ora del giorno in cui deve essere inviato il messaggio

Se non si codifica né TIME né INTERVAL, oppure se si codifica INTERVAL(0), il messaggio verrà inviato appena possibile.

E' previsto che se un messaggio non può essere spedito entro un intervallo di tempo prefissato a partire dall'ora prestabilita, esso può essere cancellato. L'intervallo di tempo viene indicato nel parametro PRGDLAY di generazione del BMS. Se tale operando

viene omesso, il CICS/VIS terra' in coda di Temporary Storage il messaggio per un tempo indefinito, fino a che non si verifichino le condizioni che ne consentano l'invio. Se invece per PRGDLAY viene specificato un valore in hhmmss, tracorso tale intervallo senza che il CICS/VIS sia riuscito a spedire il messaggio al destinario (in quanto le condizioni perche' cio' avvenga non si sono verificate), viene inviato un messaggio di errore al terminale che ha originato il messaggio.

ERRTERM (nome)

indica il nome (4 caratteri come da TCT) del terminale che deve ricevere il messaggio di errore, nel caso si desideri che esso sia diverso dal terminale che ha originato il messaggio.

OPCLASS (campo)

indica un campo di 3 byte che contiene le classi degli operatori cui il messaggio e' destinato. I bit del campo vengono associati con le 24 classi previste nella Sign-On Table nel modo seguente:

| | | | |
|--------|--------|--------|--------|
| BIT | BYTE 0 | BYTE 1 | BYTE 2 |
| 0...7 | 0...7 | 0...7 | 0...7 |
| CLASSE | 24..17 | 16...9 | 8...1 |

Se, ad esempio, un messaggio deve essere spedito agli operatori di classe 24, 16, 10, 7 e 1, il campo avra' la configurazione in bit:

100000001000001001000001

Quindi la classe viene selezionata mettendo in ON il bit che le corrisponde.

LIST (campo)

indica un campo in cui e' contenuta la lista di Routing che elenca i nomi dei terminali e/o degli operatori cui il messaggio e' indirizzato. La lista puo' avere piu' entrate, ciascuna delle quali ha il seguente tracciato:

BYTE :

- 0 - 3 Nome del terminal come da TCT, o blank (X'40').
- 4 - 5 Riservato a LDC VTAM, oppure blank (X'40').
- 6 - 8 Identificatore di operatore (OPIDENT in SNT) oppure blank (X'40').
- 9 Flag di Status imposta-

to dal BMS.

10 -15 Riservati; devono essere blank (X'40').

La fine della lista viene segnalata da due byte binari impostati a -1.

Prima che un messaggio sia spedito, devono essere soddisfatte le condizioni seguenti:

- Il terminale deve essere supportato dal BMS ed essere in servizio.
- Quanto indicato nelle opzioni TIME e INTERVAL deve essersi verificato.
- L'intervallo PRGDLAY non deve essere trascorso.

Le combinazioni dei parametri LIST e OPCLASS consentono allora al CICS/V5 di stabilire se il messaggio deve essere inviato, secondo le modalita' sotto elencate:

- Non sono stati codificati ne' LIST ne' OPCLASS. Tutti i terminali riceveranno il messaggio.
- E' stato specificato LIST ma non OPCLASS. In tal caso le entrate della lista di Routing possono:
 - contenere il nome del terminale ma non l'identificatore dell'operatore. Il messaggio verrà inviato al terminale indipendentemente dall'operatore che ha fatto SIGN-ON (se tale operazione è richiesta)
 - contenere il nome del terminale e l'identificatore dell'operatore. Il messaggio verrà inviato se e quando l'operatore farà SIGN-ON sul terminale indicato
 - contenere l'identificatore dell'operatore e non il nome del terminale. Il CICS/V5 individua, nel momento in cui viene dato il comando ROUTE, il terminale su cui l'operatore ha fatto SIGN-ON (se ha fatto SIGN-ON su piu' di un terminale, viene considerato il primo di essi elencato nella TCT) ed a tale terminale verrà inviato il messaggio solo se al momento dell'invio l'operatore non ha già fatto SIGN-OFF; se al momento in cui viene dato il comando ROUTE l'operatore in questione non ha ancora fatto SIGN-ON, l'entrata della lista di Routing viene ignorata.
- E' specificato OPCLASS ma non LIST. Il CICS/V5 ricerca tutti i terminali su cui abbia fatto SIGN-ON un operatore che appartenga ad una delle classi elencate, ed invierà ad essi il messaggio solo se al momento dell'invio l'operatore non abbia già fatto SIGN-OFF, oppure sul terminale sia presente

comunque un operatore con classe appartentente a quelle elencate.

- Sono stati codificati sia LIST che OPCLASS
 - OPCLASS ha efficacia su quelle entrate di Route List che non contengono un codice operatore, e si comporta come si è visto al punto precedente.
 - Il parametro OPCLASS viene ignorato per le entrate che contengono un codice operatore: ad essi il messaggio verrà inviato anche se la loro classe non è tra quelle elencate.

Note

Il Byte dell'entrata della lista di "Route" contenente il Flag di Status, viene impostato dal BMS dopo l'esecuzione del comando ROUTE e le configurazioni previste sono:

- X'80' Entrata ignorata
- X'40' Nome del terminale invalido
- X'20' Terminale non supportato dal BMS
- X'10' L'operatore non ha fatto SIGN-ON
- X'08' L'operatore ha fatto SIGN-ON su di un terminale non supportato dal BMS
- X'04' VTAM - LDC invalido.

TITLE (campo) indica un campo contenente un titolo che può essere visto dall'operatore al terminale usando l'opportuno comando di paginazione. Il tracciato del campo deve prevedere due byte binari per la lunghezza, seguiti dal titolo (massimo 62 byte). Nei due byte della lunghezza deve essere impostata la lunghezza totale del campo (cioè la lunghezza del titolo più due byte).

13.16.1 Gestione dell'Overflow di pagina

Non vi sono differenze, nella routine di Overflow, rispetto alla gestione di un messaggio multipagina unico: se nella lista di Routing compaiono tipi di terminali diversi, il BMS costruisce tanti messaggi logici diversi quanti sono i tipi di terminali, e per ciascun tipo, a conclusione di ogni pagina, cede il controllo alla Routine di Overflow, sempreché il programmatore abbia codificato l'apposito EXEC CICS HANDLE CONDITION OVERFLOW (label). Oltre al numero di pagina, ottenibile con comando ASSIGN, è possibile avere anche il numero dell'entrata di lista di Routing per cui si è verificato l'overflow:

EXEC CICS ASSIGN DESTCOUNT (campo)

dove "campo" va definito di due byte binari.

Si puo' utilizzare tale valore per associare al comando SEND MAP della mappa di coda eventuali campi contenenti totali parziali che, ovviamente, variano in funzione del numero di righe presenti nella pagina.

Altre informazioni sul Routing sono reperibili nel Cap. 3.2-4 dell'APRM.

13.17 SOMMARIO

Il Paging viene utilizzato per visualizzare dati di quantita' variabile, che potrebbero non essere contenuti in una sola pagina.

Per questo sono ancora utilizzate le stesse macro DFHMSD e DFHMDF, già viste nel Cap. 3, ma la macro DFHMDI presenta dei nuovi parametri per definire delle mappe di testata o di coda (HEADER e TRAILER).

L'insieme delle pagine costruite con il Paging costituiscono un "messaggio logico". Ogni pagina di questo messaggio, di solito costituita da piu' mappe (opzione ACCUM), viene memorizzata su Temporary Storage (opzione PAGING) e non sara' visualizzabile finche' il programma non chiude la costruzione del messaggio logico con un comando SEND PAGE. Solo allora potra' iniziare la sessione di paginazione del messaggio costruito, secondo le modalita' previste dalla transazione di paginazione. Dopo che il messaggio logico e' stato inviato al terminale, il programma applicativo puo' scegliere quando cancellarlo da TS ed optare se riprendere il controllo o meno.

Il Texting non fa uso di mappe ed e' utile per visualizzare dati senza un formato predefinito. Anche con il texting c'e' la possibilita' di accumulare parti di testo su una o piu' pagine. In questo caso e' possibile inserire delle informazioni di testata e/o di coda.

La stringa dei dati da "formattare", che puo' contenere anche caratteri NL e EOM inseriti dal programma, viene segmentata dal BMS in righe di dimensione congrua con quanto specificato per il terminale.

Durante l'uso del texting, l'elaborazione dell'"overflow" di pagina e' gestita automaticamente dal BMS.

Il Routing permette di trasmettere dati da uno o piu' terminali, che possono essere di tipo uguale o diverso rispetto al terminale che origina il messaggio.

Il programma applicativo identifica le destinazioni interessate alla trasmissione (cioè terminali ed operatori) definendo una lista di "routing" opportuna ed utilizzando le classi degli operatori (da 1 a 24).

Si puo' richiedere inoltre che i messaggi siano trasmessi appena possibile oppure non prima di un certo tempo.

13.18 FIGURE

| | | |
|--|-------------------------------|--------|
| Colonna----> | 1 3 5 7 9 1 1 1 1 1 2 2 2 2 2 | |
| Riga-----> | 1 3 5 7 9 1 3 5 7 9 | |
| V . | | |
| +----->1 | | PAG. 1 |
| TESTATA 2 CONTO NUMERO: 20A21/12 | | |
| o 3 INTESTATO A: MARCO PIAVINI | | |
| HEADER 4 VIA PIAVE 3 - MILANO | | |
| 5 DATA MOVIMENTO NUM.ASSIC. DARE AVERE | | |
| +----->6 | | |
| DETTAGLI 7 03/01/83 1234 100000 | | |
| w 8 10/02/83 1351 200000 | | |
| w 9 27/03/83 1476 400000 | | |
| w : : | : | : |
| w : : | : | : |
| w : : | : | : |
| w : : | : | : |
| w : : | : | : |
| w 20 20/09/83 1896 400000 | | |
| w 21 | | |
| w 22 | | |
| w 23 | | |
| CODA--->24 SALDO AL 20/09/1983 4000000 | | |
| o TRAILER ----- | | |

Figura 13.1 - Esempio di "pagina" su un terminale 24 x 80.

| | | | | | | | | | | | |
|---------|---------------------|---|---|---|---|---|---|---|---|-------|------------|
| | 1 | 2 | | | | | | | | | |
| 1 | 3 | 5 | 7 | 9 | 1 | 3 | 5 | 7 | 9 | | -->Colonna |
| Riga--> | 1 | | | | | | | | | | |
| 2 | QUESTO TESTO SERVE | | | | | | | | | | |
| 3 | | | | | | | | | | | |
| 4 | COME ESEMPIO DI USO | | | | | | | | | | |
| 5 | | | | | | | | | | | |
| 6 | DEL COMANDO TEXT | | | | | | | | | | |
| 7 | | | | | | | | | | | |
| 8 | | | | | | | | | | | |
| 9 | | | | | | | | | | | |
| : | | | | | | | | | | | |
| : | | | | | | | | | | | |
| 49 | | | | | | | | | | | |
| 50 | | | | | | | | | | | |

Figura 13.2 - Esempio di posizionamento di un testo in una pagina.

14.1 INTRODUZIONE

Questo capitolo introduce il concetto di area di memoria esterne al programma applicativo ed illustra come queste vadano definite e come sia possibile accedervi.

14.2 AREE DI MEMORIA ESTERNE

Un programma applicativo CICS puo' accedere ad aree di memoria "esterne", cioe' non facenti parte delle aree che possiede durante un'esecuzione, sfruttando le possibilita' offerte dal CICS/VS.

Le aree esterne accessibili ad un programma sono essenzialmente di due tipi:

- blocchi di controllo del CICS/VS,
- aree di input (buffer del CICS/VS)

e per conoscerne il contenuto sara' necessario, prima di tutto, definirle ed indirizzarle opportunamente.

Quando, poi, il programma in esecuzione invierà un comando per accedere all'area esterna (vedremo piu' avanti in che modo), il CICS/VS fornira' automaticamente l'aggancio all'area richiesta e ai suoi dati.

In un programma COBOL, le aree esterne devono essere definite nella LINKAGE SECTION, come strutture di dati di livello 01.

Per indirizzare queste aree il COBOL utilizza elementi di BLL (Base Locator for Linkage). In particolare, sempre nella Linkage Section, deve essere codificata, come prima definizione di livello 01, una lista di elementi di BLL, composta di tante entrate quante sono le strutture dati di livello 01 (cioe' le aree esterne), piu' una che conterra' l'indirizzo della lista stessa.

Se, per esempio, vogliamo accedere a due aree esterne, dobbiamo codificare in Linkage Section la seguente struttura:

```

LINKAGE SECTION.
 81 BLL-LISTA.
    02 FILLER      PIC S9(8) COMP.
    02 PUNTA1     PIC S9(8) COMP.
    02 PUNTA2     PIC S9(8) COMP.
 81 AREA1.

```

02 CAMPO11 PIC.....
02 CAMPO12 PIC.....

.

01 AREA2.
02 CAMPO21 PIC.....
02 CAMPO22 PIC.....

Durante l'esecuzione del programma, la prima entrata di BLL-LISTA viene caricata dal CICS/VС con l'indirizzo della lista stessa.

Come vedremo tra poco, PUNTA1 e PUNTA2 saranno impostati con gli adeguati indirizzi di memoria, se usiamo il comando ADDRESS oppure l'opzione SET nei comandi che la prevedono.

14.3 ACCESSO AI BLOCCHI DI CONTROLLO CICS/VС.

Con il comando ADDRESS il CICS/VС permette l'accesso alle seguenti aree:

- CSA (Common System Area)
- CWA (Common Work Area)
- TWA (Transaction Work Area)
- TCTUA (Terminal Control Table User Area)

Il formato del comando e'
EXEC CICS ADDRESS opzione (puntatore)

dove "opzione" e' una delle aree sopra elencate, mentre "puntatore" deve essere un elemento di BLL.

L'indirizzamento alla CSA permette l'accesso indiretto a tutti gli altri blocchi di controllo e a tutte le aree di memoria del CICS/VС.

La CWA e' usata per passare informazioni tra programmi applicativi diversi.

La TWA e' usata per passare informazioni tra programmi diversi, ma eseguiti durante lo stesso task.

La TCTUA e' usata per passare informazioni tra programmi diversi, ma eseguiti sullo stesso terminale.

Poiche' l'uso della CWA, TWA o TCTUA e' opzionale, si presenta sempre la necessita' applicativa di verificare l'esistenza, prima di chiederne l'indirizzamento col comando ADDRESS.

Per questo e' disponibile il comando ASSIGN che, tra le sue opzioni, contempla anche:

- CWALENG
- TWALENG
- TCTUALENG

cioe' permette di interrogare il CICS/VIS sulla lunghezza delle aree di CWA, TWA e TCTUA.

Il formato di questo nuovo comando e':
EXEC CICS ASSIGN opzione (area-dati).....

dove "opzione" puo' essere CWALENG, TWALENG, TCTUALENG o un qualsiasi valore tra quelli elencati nel cap. 1.6 dell'APRM; "area-dati" e' un campo di due byte binari, appositamente definito nell'area di lavoro del programma, destinato a contenere la lunghezza dell'area richiesta (CWA, TWA o TCTUA).

Di seguito possiamo vedere un esempio di utilizzo del comando ASSIGN, per ottenere la lunghezza della TWA, e del comando ADDRESS per accedere ad essa, qualora esista ($TWALENG > 0$).

WORKING-STORAGE SECTION.

77 LUNG-TWA PIC S9(4) COMP.

LINKAGE SECTION.

01 BLL-LISTA.
 02 FILLER PIC S9(8) COMP.
 02 TWA-PUNTA PIC S9(8) COMP.
01 AREA1.
 02 CAMPO11 PIC X(4).

PROCEDURE DIVISION.

 EXEC CICS ASSIGN TWALENG(LUNG-TWA)
 END-EXEC.

 IF LUNG-TWA > 0 THEN
 EXEC CICS ADDRESS TWA(TWA-PUNTA)
 END EXEC
 MOVE 'AAAA' TO CAMPO11.

14.4 ACCESSO ALLE AREE DI INPUT

Come gia' abbiamo avuto modo di dire, a proposito dei diversi comandi di lettura

- RECEIVE MAP,

- RECEIVE,
- READ, READNEXT, READPREV,
- READQ TD, READQ TS
- RETRIEVE,

quando un programma chiede di leggere dati da una mappa, un file o una coda, solitamente si richiede al CICS/Vs di collocare i dati in un'area di lavoro appositamente definita all'interno del programma (TECHICA MOVE).

Ricordiamo che quest'area-dati puo' essere la mappa logica (solo in un comando RECEIVE MAP) oppure l'argomento dell'opzione INTO.

In pratica il CICS/Vs, effettuata la lettura, prima deposita i dati in una sua area di input (buffer) e poi muove i dati nell'area di lavoro definita dal programma.

L'uso dell'opzione SET consente ad un programma di elaborare i record direttamente nel "buffer" del CICS/Vs, con conseguente maggiore rapidita' di accesso ai dati e minore dimensione della memoria di lavoro necessaria al programma.

In Cobol, l'argomento dell'opzione SET deve essere un elemento di BLL che possa essere impostato con l'indirizzo di memoria dell'area di input del CICS/Vs.

14.5 SOMMARIO

In questo capitolo abbiamo visto come accedere alle aree esterne gestite dal CICS/Vs:

- i blocchi di controllo e
- le aree di input.

Per gestire una di queste aree esterne bisogna fare due cose:

- definire una struttura dati che la rappresenti e
- definire un campo binario di quattro byte che la indirizzi.

In COBOL si usano la LINKAGE SECTION e gli elementi di BLL.

Il comando ADDRESS permette l'accesso ai blocchi di CSA, CWA, TWA e TCTUA. L'uso del comando ASSIGN consente, tra l'altro, di conoscere le dimensioni di CWA, TWA e TCTUA.

L'accesso diretto alle aree di input del CICS/Vs viene realizzato usando l'opzione SET nei vari comandi di lettura che la prevedono.

15 COMANDI PER LA GESTIONE DELLE RISORSE

15.1 INTRODUZIONE

Esistono in CICS dei comandi che controllano le transazioni che ci permettono di inviare messaggi, di interrogare/aggiornare dei files ecc.. Sono transazioni di supporto chiamate "Supplied Transactions". In questo capitolo tratteremo brevemente la transazione CEMIT. (Per ulteriori chiarimenti occorre far riferimento al manuale CICS-Supplied Transaction).

La transazione viene innescata digitando CEMIT a terminale e premendo ENTER: apparirà così l'elenco delle operazioni possibili ed in basso al video l'elenco dei tasti funzionali abilitati.

CEMIT

ENTER <----->

apparirà a video la seguente schermata:

STATUS ENTER ONE OF THE FOLLOWING

Add
Inquire
Perform
Remove
Set

APPLID=DBDICCF

PF1 HELP 3END

9MSG

15.2 COMANDI DI INTERROGAZIONE

È possibile con la transazione CEMT chiedere di visualizzare lo stato di risorse CICS; per esempio verificare lo stato di un programma CICS.
La sintassi è la seguente:

CEMT INQUIRE PROGRAM (ALZUFFP)

a seguito di questa richiesta a video comparirà:

INQUIRE PROGRAM (ALZUFFP)

STATUS: RESULTS - OVERTYPE TO MODIFY
Pro(ALZUFFP) Len(0000000) Res(000) Use(000000) Cob Ena

PF 1 HELP 3 END 7 SBH 8 SFH 9MSG 10 SB 11 SF

PRO: Nome del Programma dichiarato in PPT

LEN: Dimensione in Byte del programma (se non è ancora stato usato è zero)

RES: Quanti Task stanno usando al momento dell'INQUIRY il programma

USE: Quante volte è già stato usato il programma

COB: Appare la sigla del linguaggio in cui è scritto il programma

ENA: Stato del programma Abilitato/Disabilitato all'utilizzo

3 COMANDI DI IMPOSTAZIONE

È possibile, sempre con la transazione CEMT, "Modificare/Impostare" lo stato di risorse CICS.
È possibile per es. far conoscere al CICS la nuova versione di un programma (frutto dell'ultima compilazione) nel seguente modo:

CEMT SET PROGRAM(ALZUFFP) NEW

ENTER <---->

a seguito di tale richiesta comparirà:

SET PROGRAM (ALZUFFP) NEW

STATUS: RESULTS - OVERTYPE TO MODIFY
Pro(ALZUFFP) Len(0067240) Res(000) Use(000000) Cob Ena
NEW COPY

RESPONSE: NORMAL
PF 1 HELP 3 END 7 SBH 8 SFH 9MSG 10 SB 11 SF

NEW COPY: Significa che è stata caricata in memoria la copia/versione nuova del programma.

2001070901-16.100002.C

THIS ANNOUNCEMENT IS FOR THE RELEASE OF THE
4010 RELEASE OF THE STATE OF "GENERAL INFORMATION"
RELEASE WHICH IS 2010 TO ENHANCE THE USE OF THE
TEN CONSULTATION SERVICES WHICH ARE PROVIDED BY THE
STATE OF NEW YORK.

THE INFORMATION IS AS FOLLOWS:

1. 4010 RELEASE

2. INFORMATION RELEASES WHICH ARE PROVIDED BY

THE STATE OF NEW YORK.

3. 4010 RELEASE - STATE OF NEW YORK
AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

4. 4010 RELEASE - STATE OF NEW YORK AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

5. 4010 RELEASE - STATE OF NEW YORK AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

6. 4010 RELEASE - STATE OF NEW YORK AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

7. 4010 RELEASE - STATE OF NEW YORK AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

8. 4010 RELEASE - STATE OF NEW YORK AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

9. 4010 RELEASE - STATE OF NEW YORK AND 4010 RELEASE WHICH IS PROVIDED BY THE STATE OF NEW YORK.

16 EXECUTION DIAGNOSTIC FACILITY

16.1 INTRODUZIONE

Questo capitolo ha lo scopo di illustrarti le caratteristiche e le possibilite' di un potente strumento di debugging, che va sotto il nome di EXECUTION DIAGNOSTIC FACILITY (EDF).

Esso viene eseguito come una transazione CICS/V5 e viene attivato mediante il codice CEDF, oppure mediante la pressione di un tasto PFxx opportunamente associato in PCT a EDF.

Richiede schermi 3270 da 24 righe per 80 colonne, o piu' grandi, con o senza tasti funzionali.

La manovra delle operazioni di debugging viene eseguita mediante i tasti ENTER o PFxx, ed ogni pannello di EDF e' autoesplorativo, poiche' porta nelle ultime 5 linee in basso dello schermo le funzioni associate a ciascun tasto.

Qualora la tastiera 3270 sia priva di tasti funzionali, le funzioni associate ad un tasto PF vengono eseguite ponendo il cursore sotto la stringa di caratteri che illustra la funzione del particolare PF, e agendo poi sul tasto ENTER (selezione mediante cursore).

16.2 COME INIZIARE UNA SESSIONE DI DEBUGGING

Si attiva EDF su di un terminale introducendo il codice di transazione CEDF, e premendo ENTER (oppure, come gia' detto, si fa uso di un tasto PF associato in PCT alla transazione CEDF).

Il terminale e' ora pronto per il debugging: basta introdurre il codice della transazione di cui si vuol eseguire il test, e la transazione in questione verrà eseguita sotto il controllo di EDF, il quale intercetta l'esecuzione del (dei) programma (programmi) nei punti seguenti:

- All'inizio della transazione, dopo che il blocco EIB e' stato inizializzato, ma prima che il programma applicativo abbia ricevuto il controllo.

Sullo schermo del terminale viene riportato il blocco EIB con i suoi vari campi.

- Prima dell'esecuzione di ogni comando EXEC.

Sullo schermo del terminale viene riportato il comando, con le sue parole chiave, gli argomenti, e le opzioni usate.

Al comando viene anche associato l' offset esadecimale all' interno del programma, o, se il programma e' stato tradotto con la opzione DEBUG, il numero di linea assegnato ad esso nel listing del traduttore di comandi.

- Dopo l' esecuzione di ogni comando, prima che il meccanismo di HANDLE CONDITION eventualmente associato al comando venga attivato.

In questo caso, oltre al comando, viene evidenziato sullo schermo il tipo di completamento (ad es.: NORMAL, MAPFAIL, ecc.).

- Alla fine di ogni programma che venga eseguito per il task.
- Alla fine normale del task.
- Al verificarsi di un ABEND.

Nel caso di codice di abend ASRA (program check), viene anche fornito l' offset (o numero di linea, vedi punto 2 precedente) della istruzione in errore.

In ogni momento e' possibile esaminare:

- EIB e DIB (DL/I Interface Block),
- Area di lavoro del programma,
- Gli ultimi dieci (10) comandi eseguiti,
- Il contenuto di qualunque area di memoria all' interno della regione CICS/VS.

16.3 POSSIBILITÀ DI MODIFICA.

Quando EDF intercetta l' esecuzione in uno dei punti precedentemente visti, e' possibile alterare temporaneamente (cioe' solo per quella esecuzione):

- Gli ARGOMENTI di un comando che sta per essere eseguito. E' anche possibile sopprimere l' esecuzione del comando, sostituendo il comando medesimo con la parola NOOP (ad es. SEND venga cambiato in NOOP).
- La risposta all'esecuzione di un comando, sempre per sovra-scrittura. Se quindi una READ e' stata eseguita bene, e la risposta risultante e' NORMAL, e' possibile andare a testare la routine di gestione delle condizioni NOTFND sostituendo a NORMAL la parola NOTFND, ed attivando quindi la HANDLE CONDITION relativa.

- I campi di EIB e DIB.

Come si vede, le possibilita' di "seguire", comando per comodato il flusso di un task, e di intervenire con modifiche temporanee per alterare il flusso medesimo, fanno di EDF uno strumento molto valido.

Oltre alle capacita' di modifiche temporanee appena viste, l'utente puo' sopprimere la presentazione dei pannelli EDF fino a che il flusso non arrivi alla esecuzione di un prestabilito comando.

L' elenco completo e dettagliato di tutte le possibilta' operative di EDF, e le loro spiegazioni, e' fornita nel Cap. 1.7 dell' APRM.

16.4 ESEMPIO DI USO DELL'EDF.

Qui di seguito, viene dato un esempio di sessione EDF che chiarisce come si possa eseguire il debugging di una transazione.

Per questo utilizzeremo i programmi di Inquiry/Update che hai completato durante le esercitazioni al Personal Computer.

Dopo aver richiamato l' EDF ed aver introdotto il codice di transazione TC10, i pannelli EDF cominciano apresentarsi cosi':

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001042 DISPLAY: 00
STATUS: PROGRAM INITIATION

EIBTIME = +0150104
EIBDATE = +0083319
EIBTRNID = 'TC10'
EIBTASKN = +0001042
EIBTRMID = 'TERF'
EIBCPOSN = +00005
EIBCALEN = +00000
EIBAID = X'7D'
EIBFN = X'0000'
EIBRCODE = X'000000000000'
EIBDS = '.....'
EIBREQID = '.....'
+ EIBRSRCE = ''

AT X'00931C52'
AT X'00931C53'
AT X'00931C55'

RESPONSE:

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: UNDEFINED

Dopo aver attivato EDF e introdotto il codice di transazione (TC10, in questo caso), questo e' il primo pannello che viene presentato: EIB con i suoi vari campi, cosi' come sono stati impostati all'inizio del task. Dopo aver esaminato EIB, per continuare basta premere il tasto di ENTER, come ti avverte la quinultima riga.

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001042 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS HANDLE CONDITION
MAPFAIL()

OFFSET:X'000814' EIBFN=X'0204'
RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

Come vedi, il primo comando eseguito dal programma e' questo. Esso sta per essere eseguito. Per mandarlo in esecuzione, devi premere ENTER. Fatto questo, ti viene mostrato il pannello seguente...

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001042 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS HANDLE CONDITION
MAPFAIL()

OFFSET:X'000814'
RESPONSE: NORMAL

EIBFN=X'0204'
EIBRCODE=X'000000000000'

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

Il comando HANDLE CONDITION e' stato eseguito correttamente,
infatti la risposta e' NORMAL. Se premi ora ENTER, EDF arrestera'
l'esecuzione del tuo programma al successivo comando, che e'..

```
TRANSACTION: TC10 PROGRAM: PGMC0BA0 TASK NUMBER: 0001042 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
MAP('MAPPA0 ')
FROM('.....')
MAPSET('MAPSETE')
TERMINAL
ERASE

OFFSET:X'0008AC'
RESPONSE: EIBFM=X'1804'
REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Note che qui c' e' un errore: il nome della mappa e' MAPPA0, e non MAPPA0. Se lasci eseguire il comando, agendo, come ormai hai imparato, sul tasto ENTER, ecco cosa accade...

TRANSACTION: TC10 PROGRAM: PGMCOBA0 TASK NUMBER: 0001042 DISPLAY: 00
STATUS: AN ABEND HAS OCCURRED

| | | |
|------------|-----------------------|----------------|
| EIBTIME | = +0150104 | |
| EIBDATE | = +0083319 | |
| EIBTRNID | = 'TC10' | |
| EIBTASKN | = +0001042 | |
| EIBTRMID | = 'TERF' | |
| EIBCPOSH | = +00005 | |
| EIBCALEN | = +00000 | |
| EIBAID | = X'7D' | AT X'00931C52' |
| EIBFN | = X'1804' SEND | AT X'00931C53' |
| EIBRCODE | = X'0000000000000000' | AT X'00931C55' |
| EIBDS | = '.....' | |
| EIBREQID | = '.....' | |
| + EIBRSRCE | = ' | |

ABEND : ABM0

REPLY:

ENTER: CONTINUE

| | | |
|-------------------------|-----------------------|-----------------------|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: UNDEFINED |

La SEND MAP si e' completata con abend di codice ABM0. Il manuale CICS/VS Messages and Codes, recita per questo codice: la mappa specificata per la richiesta di BMS non puo' essere localizzata. Infatti, abbiamo gia' osservato che il nome fornito nella SEND MAP e' errato. A questo punto il task termina anormalmente, basta premere ENTER...

TRANSACTION: TC10 PROGRAM:
STATUS: ABNORMAL TASK TERMINATION

TASK NUMBER: 0001042 DISPLAY: 00

| | | | |
|------------|-------------------|--|----------------|
| EIBTIME | = +0150104 | | |
| EIBDATE | = +0083319 | | |
| EIBTRNID | = 'TC10' | | |
| EIBTASKN | = +0001042 | | |
| EIBTRMID | = 'TERF' | | |
| EIBCPOSH | = +00005 | | |
| EIBCALEN | = +00000 | | |
| EIBAID | = X'7D' | | |
| EIBFN | = X'1804' SEND | | AT X'80931C52' |
| EIBRCODE | = X'000000000000' | | AT X'80931C53' |
| EIBDS | = '.....' | | AT X'80931C55' |
| EIBREQID | = '.....' | | |
| + EIBRSRCE | = '.....' | | |

ABEND : ABM0

TO CONTINUE EDF SESSION REPLY YES

REPLY: no

ENTER: CONTINUE

| | | |
|-------------------------|-----------------------|-----------------------|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: UNDEFINED |

Nota qui (sestultima riga), che EDF ti chiede se vuoi continuare la sessione o no: cio' accade ogni volta che termina un TASK. Se rispondi "no" (che e' la risposta che EDF ti imposta dopo la parola chiave REPLY), il terminale torna in stato normale. Immaginiamo invece di rispondere "yes", in questo modo...

TRANSACTION: TC10 PROGRAM:
STATUS: ABNORMAL TASK TERMINATION

TASK NUMBER: 0001042 DISPLAY: 00

| | | |
|------------|-------------------|----------------|
| EIBTIME | = +0150104 | |
| EIBDATE | = +0083319 | |
| EIBTRNID | = 'TC10' | |
| EIBTASKN | = +0001042 | |
| EIBTRMID | = 'TERF' | |
| EIBCPOSN | = +00005 | |
| EIBCALEN | = +00000 | |
| EIBAID | = X'7D' | AT X'00931C52' |
| EIBFN | = X'1804' SEND | AT X'00931C53' |
| EIBRCODE | = X'000000000000' | AT X'00931C55' |
| EIBDS | = '.....' | |
| EIBREQID | = '.....' | |
| + EIBRSRCE | = ' | |

ABEND : ABM0
TO CONTINUE EDF SESSION REPLY YES
ENTER : CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: UNDEFINED

REPLY: yes

Come vedi si e' semplicemente scritto SOPRA a "no" la parola "yes". Premendo ora ENTER, compare sul terminale il messaggio di errore che, in ogni caso, il CICS/V5 avrebbe inviato, anche se la transazione NON fosse stata eseguita sotto il controllo di EDF...

tc10 DFH2005I TRANSACTION TC10 ABEND ABM0 IN PROGRAM PGMCOBAD 15:02:41

Vogliamo ora rimediare all' errore che abbiamo rilevato. Siamo ancora sotto controllo di EDF, perciò, introduciamo il codice di transazione, dopo aver pulito lo schermo con il tasto CLEAR..

tc10

Diamo ENTER....

...ma l'invio della linea può essere fatto anche con una
frequenza di invio di circa 2000 ms. In effetti non avremo
bisogno di una certa attesa per ricevere i dati.

TRANSACTION: TC10 PROGRAM: PGMCBA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: PROGRAM INITIATION

| | | |
|------------|---------------------|----------------|
| EIBTIME | = +0150256 | |
| EIBDATE | = +0083319 | |
| EIBTRNID | = 'TC10' | |
| EIBTASKH | = +0001071 | |
| EIBTRMID | = 'TERF' | |
| EIBCPOSH | = +00004 | |
| EIBCALEN | = +00000 | |
| EIBAID | = X'7D' | AT X'00931C52' |
| EIBFM | = X'0000' | AT X'00931C53' |
| EIBRCODE | = X'00000000000000' | AT X'00931C55' |
| EIBDS | = '.....' | |
| EIBREQID | = '.....' | |
| + EIBRSRCE | = ' | |

RESPONSE:

REPLY:

ENTER: CONTINUE

| | | |
|-------------------------|-----------------------|-----------------------|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: UNDEFINED |

Ricominciano ad apparire i pannelli di EDF: diamo enter e proseguiamo.

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS HANDLE CONDITION
MAPFAIL()

OFFSET:X'000814'
RESPONSE:

EIBFM=X'0204'

REPLY:

ENTER CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

Ancora ENTER.....

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS HANDLE CONDITION
MAPFAIL()

OFFSET:X'000814'
RESPONSE: NORMAL EIBFN=X'0204'
EIBRCODE=X'000000000000'
REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

ENTER.....

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
MAP('MAPPAD ')
FROMC'.....'
MAPSET('MAPSETE')
TERMINAL
ERASE

OFFSET:X'0008AC' EIBFN=X'1804'
RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF1 : UNDEFINED | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF4 : SUPPRESS DISPLAYS | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF7 : SCROLL BACK | PF11: UNDEFINED | PF12: ABEND USER TASK |
| PF10: PREVIOUS DISPLAY | | |

Eccoci al comando in errore. Dobbiamo trasformare MAPPAD (un argomento, infatti sta tra parentesi!), in MAPPA0, perciò sposteremo il cursore sopra MAPPA0, e scriveremo...

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
MAP('MAPPAD')
FROM('.....')
MAPSET('MAPSETE')
TERMINAL
ERASE

OFFSET:X'0008AC' EIBFN=X'1804'
RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

Ora il comando e' a posto. Infatti, se premi il solito ENTER.....

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

CHIAVE DI RICERCA

I INQUIRY

U UPDATE

F FINE

Come vedi, sullo schermo compare il pannello di MENU che ben conosci. Non introdurre ora alcun dato, ma limitati ad agire sul tasto ENTER...

```
TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND
MAP('MAPPAD')
FROM('.....')
MAPSET('MAPSETE')
TERMINAL
ERASE
```

```
OFFSET:X'0008AC' EIBFN=X'1804'
RESPONSE: NORMAL EIBRCODE=X'000000000000'
REPLY:
```

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF1 : UNDEFINED | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF4 : SUPPRESS DISPLAYS | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF7 : SCROLL BACK | PF11: UNDEFINED | PF12: ABEND USER TASK |
| PF10: PREVIOUS DISPLAY | | |

Il comando SEND MAP e' stato eseguito correttamente (lo sapevamo già: abbiamo visto sullo schermo il pannello di MENU!), perciò andiamo avanti, con il solito ENTER...

```
TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS RECEIVE
MAP('MAPPA0 ')
INTO('.....')
MAPSET('MAPSETE')
TERMINAL
```

OFFSET:X'00090C' EIBFN=X'1802'
RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF1 : UNDEFINED | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF4 : SUPPRESS DISPLAYS | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF7 : SCROLL BACK | PF11: UNDEFINED | PF12: ABEND USER TASK |
| PF10: PREVIOUS DISPLAY | | |

Ecco il prossimo comando: RECEIVE di MAPPA0. Osserva che l' area
in cui verranno letti i dati non contiene nulla. Premi ENTER, e
sullo schermo ritorna il MENU...

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

CHIAVE DI RICERCA

I INQUIRY

U UPDATE

F FINE

RICERCA PER COD. 1000
TAJHEED, REED 1 000
EROTICHOUS, ROSTA 1 000
DEAT, REED, REED 1 000

RANDONNEY, ROBIN 1 000
ROBUSTO, SHIRLEY 1 000
CHAMBERLAIN, JAMES 1 000
DUNN, ROBERT 1 000

SIMONE, LORENZO
SEKEMBO, 1 000
REAGAN, ROSEMARY 1 000
POLE, JORDAN 1 000

Supponi di voler provare se la routine di gestione della condizione di MAPFAIL funziona bene oppure no. In questo caso, senza introdurre alcun dato e pigiando ENTER, provocherai appunto un MAPFAIL...

```
TRANSACTION: TC10 PROGRAM: PGMCBA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS RECEIVE
MAP('MAPPA0 ')
INTO('.....')
MAPSET('MAPSETE')
TERMINAL
```

```
OFFSET:X'00090C'
RESPONSE: MAPFAIL
EIBFM=X'1802'
EIBRCODE=X'040000000000'
REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Infatti. Vediamo ora cosa accade: altro ENTER...

```
TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00  
STATUS: ABOUT TO EXECUTE COMMAND  
EXEC CICS SEND  
MAP('MAPPA0 ')  
FROM('.....*** FUNZIONE ERRATA *** .....')  
MAPSET('MAPSETE')  
CURSOR(-00001)  
TERMINAL
```

OFFSET:X'0009F6'
RESPONSE: EIBFN=X'1804'

REPLY:

| | | | | | | |
|------------------------|-------------------------|-----------------------|--------------------|-------------------|----------------------|-----------------------|
| ENTER: CONTINUE | PF2 : SWITCH HEX/CHAR | | | PF3 : UNDEFINED | | |
| PF1 : UNDEFINED | PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY | PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK | | | | |

Bene: la routine di gestione del MAPFAIL funziona, infatti, come vedi, il programma ha preparato il messaggio previsto allo scopo, e si appresta ad inviarlo. Da qui in avanti (come per tutte le manovre che hai gia' visto) verrà usato sempre e solo il tasto ENTER, quindi non ti verrà più evidenziato di volta in volta: limitati a scorrere via via le pagine considerandole come una successione di pannelli su video, come hai fatto del resto finora.

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

CHIAVE DI RICERCA

I INQUIRY

U UPDATE

F FINE PROVA

***** FUNZIONE ERRATA *****

La segnalazione di errore e' giunta al terminale.

Per ulteriori informazioni consultare la sezione di riferimento al capitolo sulle funzioni di ricerca ed inserire le chiavi di ricerca nel campo ricerca della finestra delle funzioni. Per ulteriori informazioni sulla funzione di ricerca si consiglia di leggere il manuale CICS Reference Guide.

```
TRANSACTION: TC10 PROGRAM: PGMCBA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND
MAP('MAPPAC ')
FROMC('.....*** FUNZIONE ERRATA ***')
MAPSET('MAPSETE')
CURSOR(-00001)
TERMINAL
```

```
OFFSET:X'0009F6'
RESPONSE: NORMAL EIBFN=X'1804'
EIBRCODE=X'0000000000000000' REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

La SEHD MAP si e' completata evidentemente bene.

```
TRANSACTION: TC10    PROGRAM: PGMCBA0    TASK NUMBER: 0001071    DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS RECEIVE
MAP('MAPPA0 ')
INTO('.....*** FUNZIONE ERRATA ***')
MAPSET('MAPSETE')
TERMINAL
```

OFFSET:X'00090C' EIBFN=X'1802'
RES INSE:

REPLY:

| | | |
|------------------------|-----------------------|-----------------------|
| ENTE : CONTINUE | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF1 UNDEFINED | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF4 SUPPRESS DISPLAYS | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF7 : SCROLL BACK | PF11: UNDEFINED | PF12: ABEND USER TASK |
| PF10: PREVIOUS DISPLAY | | |

Ora il programma si appresta ad eseguire la RECEIVE MAP. Nota che nell' area della mappa vi è ancora il messaggio di errore preparato dalla routine di MAPFAIL.

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

CHIAVE DI RICERCA

I INQUIRY

U UPDATE

F FINE

*** FUNZIONE ERRATA ***

Introduciamo ora i dati, per esempio...

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

i

CHIAVE DI RICERCA

01

I INQUIRY

U UPDATE

F FINE

*** FUNZIONE ERRATA ***

.....in questo modo.

```
TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS RECEIVE
MAP('MAPPAD')
INTO('.....I...01.....')
MAPSET('MAPSETE')
TERMINAL
```

```
OFFSET:X'00090C'
RESPONSE: NORMAL
EIBFN=X'1802'
EIBRCODE=X'0000000000000000'
REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Ed ecco che la RECEIVE si completa, ed i dati sono nell' area della mappa.

```
TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS XCTL
PROGRAM('PGMCODE0')
COMMAREA('I01')
LENGTH(+00003)
```

OFFSET:X'000A52' EIBFN=X'0E04'
RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

Il programma PGMCOBAD sta per cedere il controllo al programma di
INQUIRY PGMCODE0.

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: PROGRAM TERMINATION

RESPONSE:

ENTER: CONTINUE

| | | |
|-------------------------|-----------------------|-----------------------|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

REPLY:

PGMCOBAD e' terminato: ha ceduto il controllo a PGMCOBED, infatti..

TRANSACTION: TC10 PROGRAM: PGMCOBEO TASK NUMBER: 0001071 DISPLAY: 00
 STATUS: PROGRAM INITIATION
 COMMAREA = 'I01...X.....Z.....'
 EIBTIME = +0150256
 EIBDATE = +0083319
 EIBTRNID = 'TC10'
 EIBTASKN = '+0001071'
 EIBTRMID = 'TERF'
 EIBCPOSH = +00280
 EIBCALEN = +00003
 EIBAID = X'7D'
 EIBFN = X'0E04' XCTL
 EIBRCODE = X'000000000000'
 EIBDS = '.....'
 EIBREQID = '.....'
 + EIBRSRCE = ' '

RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: UNDEFINED |

Osserva cosa contiene la COMMAREA! Tutto funziona. Potresti alterare pero' il contenuto della COMMAREA (scrivendo sopra, semplicemente!), se tu volessi.

TRANSACTION: TC10 PROGRAM: PGMCOBEDO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS HANDLE AID
CLEAR()
ANYKEY()

OFFSET:X'001056' EIBFN=X'0206'
RESPONSE:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

REPLY:

TRANSACTION: TC10 PROGRAM: PGMCODE0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS HANDLE AID
CLEAR()
ANYKEY()

OFFSET:X'001056'
RESPONSE: NORMAL

EIBFN=X'0206'
EIBRCODE=X'000000000000'

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

TRANSACTION: TC10 PROGRAM: PGMCODE00 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC DLI SCHEDULE
PSB(PSBFORC)

OFFSET:X'001EC8' LINE:00267 EIBFN=X'0004'

RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

Questo e' il comando di schedulazione del PSB DL/I.

TRANSACTION: TC10 PROGRAM: PGMCOBEO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI SCHEDULE
PSB(PSBFORC)

OFFSET:X'001EC8' LINE:00267 EIBFM=X'0004'
RESPONSE: ''

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

- Il responso e' '' (X'4040'), perciò tutto e' OK: il PSB e' schedulato e si puo' accedere al Data Base.

TRANSACTION: TC10 PROGRAM: PGMCOBED TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC DLI GET UNIQUE
USING PCB(+00001)

SEGMENT(FORNITOR)
INTO('.....')
SEGLENGTH(+00072)
WHERE(CDDFORM = '01')
FIELDLENGTH(+00002)

OFFSET:X'0021BE' LINE:00327 EIBFN=X'000A'
RESPONSE:

| | | | |
|-------------------------|-----------------------|-----------------------|--|
| ENTER: CONTINUE | REPLY: | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED | |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY | |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS | |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK | |

Ora si sta per leggere il Fornitore di codice 01, come richiesto.

```
TRANSACTION: TC10 PROGRAM: PGMCOBED TASK NUMBER: 0001071 DISPLAY: 00  
STATUS: COMMAND EXECUTION COMPLETE  
EXEC DLI GET UNIQUE  
USING PCB(+00001)
```

```
SEGMENT(FORNITOR)  
INTO('01 ROSSINI')  
SEGLLENGTH(+00072)  
WHERE(CODFORN = '01')  
FIELDLENGTH(+00002)
```

VIA VALLEAMBROSIA 11 ASSAGO MI

```
OFFSET:X'0021BE' LINE:00327 EIBFN=X'000A'  
RESPONSE: ''
```

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF1 : UNDEFINED | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF4 : SUPPRESS DISPLAYS | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF7 : SCROLL BACK | PF11: UNDEFINED | PF12: ABEND USER TASK |
| PF10: PREVIOUS DISPLAY | | |

E nell' area di I/O del programma, come vedi, ecco il Fornitore richiesto.

TRANSACTION: TC10 PROGRAM: PGMCODEO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
MAP('MAPPA1')
FROM('.....01...ROSSINI')
MAPSET('MAPSETE')
TERMINAL
ERASE

INIZIO DI ATTIVAZIONE AREA DI MAPPAGGIO

OFFSET:X'801126'
RESPONSE:

EIBFN=X'1804'

| ENTER: CONTINUE | REPLY: | | |
|-------------------------|-----------------------|-----------------------|--|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED | |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY | |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS | |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK | |

Mossi i dati da area di I/O del Data Base ad area della mappa,
essi vengono inviati al terminale.

** ANAGRAFICA FORNITORE **

CODICE F. 01

COGNOME E NOME ROSSINI

INDIRIZZO VIA VALLEAMBROSIA 11 ASSAGO MI

PER CONTINUARE BATTI CLEAR

.....

TRANSACTION: TC10 PROGRAM: PGMCODEO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND
MAP('MAPPA1')
FROM('.....01...ROSSINI') ...VIA VALLEAMBROSIA '...)
MAPSET('MAPSETE')
TERMINAL
ERASE

OFFSET:X'001126'
RESPONSE: 'NORMAL' EIBFN=X'1804'
EIBRCODE=X'000000000000'
REPLY:
ENTER: CONTINUE PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF1 : UNDEFINED PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

Il comando SEND MAP e' andato bene.

TRANSACTION: TC10 PROGRAM: PGMCODE0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS RECEIVE

OFFSET=X'00114C'
RESPONSE:

EIBFM=X'0402'

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

Ed ora il programma aspetta il CLEAR per continuare.

** ANAGRAFICA FORNITORE **

CODICE 01

COGNOME E NOME ROSSINI

INDIRIZZO VIA VALLEAMBROSIA 11 ASSAGO MI

PER CONTINUARE BATTI CLEAR

Supponi di premere il CLEAR.

TRANSACTION: TC10 PROGRAM: PGMCOBEO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS RECEIVE

OFFSET:X'00114C'
RESPONSE: NORMAL
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

EIBFN=X'0402'
EIBRCODE=X'000000000000'

REPLY:

- Sei sicuro di aver premuto il CLEAR? Puoi accertarlo in due modi:

- Premere PF5 ed avere sullo schermo l'area di lavoro del programma
- Premere quindi PF4, sul pannello che mostra l'area di lavoro, per evidenziare l'EIB (come all'inizio del programma sotto EDF) e verificare il contenuto del campo EIBAID, che contiene la configurazione esadecimale del tasto premuto.

Oppure, che e' piu' semplice, premere ENTER e vedere cosa accade...

TRANSACTION: TC10 PROGRAM: PGMCODEO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC DLI TERMINATE

OFFSET:X'000FC0' LINE:00076 EIBFN=X'0006'

RESPONSE:

REPLY:

| | | | |
|-------------------------|-----------------------|-----------------------|--|
| ENTER: CONTINUE | | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED | |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY | |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS | |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK | |

Venne eseguito il comando di terminazione DL/I: e' stato in effetti premuto CLEAR.....

TRANSACTION: TC10 PROGRAM: PGMC0BEO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI TERMINATE

OFFSET:X'000FC0' LINE:00076 EIBFM=X'0006'
RESPONSE: '' REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

Ora il PSB e' stato rilasciato, e.....

TRANSACTION: TC10 PROGRAM: PGMCOBED TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS XCTL
PROGRAM('PGMCOBAD')

OFFSET:X'001026' EIBFN=X'0E04'
RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

Si torna, come previsto, al programma PGMCOBAD.

TRANSACTION: TC10 PROGRAM: PGMC0B0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: PROGRAM TERMINATION

RESPONSE:

ENTER: CONTINUE PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

REPLY:

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: PROGRAM INITIATION

| | | |
|------------|-------------------|----------------|
| EIBTIME | = +0150256 | |
| EIBDATE | = +0083319 | |
| EIBTRNID | = 'TC10' | |
| EIBTASKN | = +0001071 | |
| EIBTRMID | = 'TERF' | |
| EIBCPSON | = +00000 | |
| EIBCALEN | = +00000 | |
| EIBAID | = X'7D' | AT X'00931C52' |
| EIBFN | = X'0E04' XCTL | AT X'00931C53' |
| EIBRCODE | = X'000000000000' | AT X'00931C55' |
| EIBDS | = '.....' | |
| EIBREQID | = '.....' | |
| + EIBRSRCE | = '.....' | |

RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: UNDEFINED |

Siamo nuovamente all'inizio del programma PGMCOBAD.

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS HANDLE CONDITION
MAPFAIL()

02156647X TA
02156648X TA
02156649X TA

OFFSET:X'000814'
RE:ONSE:

EIBFH=X'0204'

REPLY:

ENTE?: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS HANDLE CONDITION
MAPFAIL()

OFFSET=X'900814'
RESPONSE: NORMAL

EIBFN=X'0204'
EIBRCODE=X'00000000000000'

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

```
TRANSACTION: TC10 PROGRAM: PGMC0BA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
MAP('MAPPA0')
FROM('.....')
MAPSET('MAPSETE')
TERMINAL
ERASE
```

```
OFFSET:X'0008AC' EIBFN=X'1804'
RESPONSE:
REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK
```

Qui occorre ancora correggere MAPPA0 in MAPPA0! Non dimenticare che le variazioni sono temporanee!

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
MAPC('MAPPA0')
FROM('.....')
MAPSETC('MAPSETE')
TERMINAL
ERASE

OFFSET:X'0008AC' EIBFH=X'18D4'
RESPONSE:

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

CHIAVE DI RICERCA.

I INQUIRY

U UPDATE

F FINE

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND
MAP('MAPPAD ')
FROMC'.....'
MAPSET('MAPSETE')
TERMINAL
ERASE

OFFSET:X'0008AC'
RESPONSE: NORMAL

EIBFN=X'1804'
EIBRCODE=X'000000000000'

REPLY:

| | | | |
|-------------------------|-----------------------|-----------------------|--|
| ENTER: CONTINUE | | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION | |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY | |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS | |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK | |

TRANSACTION: TC10 PROGRAM: PGMCBA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS RECEIVE
MAP('MAPPA0 ')
INTO('...')
MAPSET('MAPSETE')
TERMINAL

OFFSET:X'00090C'

EIBFN=X'1802'

RESPONSE:

REPLY:

ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

CHIAVE DI RICERCA

I INQUIRY

U UPDATE

F FINE

Introduciamo F per chiudere....

MAPPA DI INQUIRY/UPDATE

CODICE FUNZIONE

f

CHIAVE DI RICERCA

..

I INQUIRY

STAZIONE TO AVANTO

U UPDATE

STAZIONE TO AVANTO

F FINE

STAZIONE ..

STAZIONE ..

STAZIONE ..

TRANSACTION: TC10 PROGRAM: PGMCBA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS RECEIVE
MAP('MAPPA0')
INTO('.....F.....')
MAPSET('MAPSETE')
TERMINAL

OFFSET:X'00090C'
RESPONSE: NORMAL EIBFN=X'1802'
EIBRCODE=X'000000000000'
REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND' USER TASK

La RECEIVE e' andata bene, ed F e' stato ricevuto, come vedi!

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS SEND
FROM(' * FINE TRANSAZIONE * ')
LENGTH(+00020)
ERASE

OFFSET:X'000AE2'
RESPONSE:

EIBFM=X'0404'

REPLY:

ENTER: CONTINUE

| | | |
|-------------------------|-----------------------|-----------------------|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

■ FINE TRANSAZIONE ■

Il messaggio di Fine Transazione arriva sullo schermo.....

TRANSACTION: TC10 PROGRAM: PGMCBA0 TASK NUMBER: 0001071 DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND
FROM(' * FINE TRANSAZIONE *')
LENGTH(+00020)
ERASE

OFFSET=X'000AE2'
RESUME: NORMAL EIBFN=X'0404'
EIBRCODE=X'000000000000'
REPLY:
ENTER: CONTINUE
PF1 : UNDEFINED PF2 : SWITCH HEX/CHAR PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS PF5 : WORKING STORAGE PF6 : USER DISPLAY
PF7 : SCROLL BACK PF8 : SCROLL FORWARD PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY PF11: UNDEFINED PF12: ABEND USER TASK

00 1A5F00 11500000 00000000 00000000 00000000 00000000

TRANSACTION: TC10 PROGRAM: PGMCOBAD TASK NUMBER: 0001071 DISPLAY: 00
STATUS: ABOUT TO EXECUTE COMMAND
EXEC CICS RETURN

OFFSET:X'000B08'

EIBFN=X'0E08'

RESPONSE:

REPLY:

| | | |
|-------------------------|-----------------------|-----------------------|
| ENTER: CONTINUE | | |
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : UNDEFINED |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: ABEND USER TASK |

Ed il Task chiude.

10000 - 0000 SYSTEM AT 1000000000 - 0000 0000

TRANSACTION: TC10 PROGRAM: PGMCDBAO TASK NUMBER: 0001071 DISPLAY: 00
STATUS: PROGRAM TERMINATION

RESPONSE:

ENTER: CONTINUE

PF1 : UNDEFINED

PF6 : SUPPRESS DISPLAYS

PF7 : SCROLL BACK

PF10: PREVIOUS DISPLAY

PF2 : SWITCH HEX/CHAR

PF5 : WORKING STORAGE

PF8 : SCROLL FORWARD

PF11: UNDEFINED

PF3 : END EDF SESS

PF6 : USER DISPLAY

PF9 : STOP CONDITIONS

PF12: ABEND USER TASK

TRANSACTION: TC10 PROGRAM:
STATUS: TASK TERMINATION

TASK NUMBER: 0001071 DISPLAY: 00

RESPONSE:

TO CONTINUE EDF SESSION REPLY YES
ENTER: CONTINUE

REPLY: NO

| | | |
|-------------------------|-----------------------|-----------------------|
| PF1 : UNDEFINED | PF2 : SWITCH HEX/CHAR | PF3 : END EDF SESSION |
| PF4 : SUPPRESS DISPLAYS | PF5 : WORKING STORAGE | PF6 : USER DISPLAY |
| PF7 : SCROLL BACK | PF8 : SCROLL FORWARD | PF9 : STOP CONDITIONS |
| PF10: PREVIOUS DISPLAY | PF11: UNDEFINED | PF12: UNDEFINED |

Il task e' finito, e non si vuole continuare con EDF (come puoi vedere in REPLY=NO). Dunque, sullo schermo ti rimane ora.....

■ ETNE TRANSAZIONE ■

La Sessione EDF s' è conclusa.

Dovresti a questo punto aver apprezzato la comodita' e l'affidabilita' dell' EDF. Un poco di pratica su un sistema reale bastara', dopo lo studio dell'esempio appena completato, a renderti perfettamente padrone di questo importante strumento di debuggaggio.