

# ***z/OS JOB CONTROL LANGUAGE***

**FIFTH EDITION**

---

*Gary DeWard Brown*

**WILEY COMPUTER PUBLISHING**



*John Wiley & Sons, Inc.*



# ***z/OS JOB CONTROL LANGUAGE***

**FIFTH EDITION**

---

*Gary DeWard Brown*

**WILEY COMPUTER PUBLISHING**



*John Wiley & Sons, Inc.*

Publisher: Robert Ipsen  
Editor: Margaret Eldridge  
Developmental Editor: Kathryn A. Malm  
Associate Managing Editor: Penny Linskey  
New Media Editor: Brian Snapp  
Text Design & Composition: North Market Street Graphics

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where John Wiley & Sons, Inc., is aware of a claim, the product names appear in initial capital or ALL CAPITAL LETTERS. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This book is printed on acid-free paper. ∞  
Copyright © 2002 by Gary DeWard Brown. All rights reserved.

Published by John Wiley & Sons, Inc.,  
Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4744. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, (212) 850-6011, fax (212) 850-6008, E-Mail: PERMREQ@WILEY.COM.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold with the understanding that the publisher is not engaged in professional services. If professional advice or other expert assistance is required, the services of a competent professional person should be sought.

Library of Congress Cataloging-in-Publication Data:  
ISBN 0471-236357

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

---

# ***CONTENTS***

<b><i>Preface</i></b>	<b><i>ix</i></b>
<b><i>Job Control Language Parameters</i></b>	<b><i>xi</i></b>
<b><i>Chapter 1 Introduction</i></b>	<b><i>1</i></b>
1.1 The Shock of JCL	1
1.2 The Role of JCL	3
1.3 The Difficulty of JCL	3
1.4 The Approach to JCL	4
<b><i>Chapter 2 Introduction to JCL and z/OS</i></b>	<b><i>6</i></b>
2.1 z/OS Concepts and Vocabulary	6
2.2 z/OS Hardware Architecture	10
2.3 Computer Data	26
2.4 Data Storage	33
<b><i>Chapter 3 JCL within a Job</i></b>	<b><i>35</i></b>
3.1 JCL Statements	35
3.2 Computer Jobs	36
3.3 Creating Programs	41
3.4 Sort Example	42
3.5 Compile, Linkage Edit, Execute Example	45
3.6 Cataloged Procedure	50
<b><i>Chapter 4 JCL Statement Formats and Rules</i></b>	<b><i>51</i></b>
4.1 JCL Statement Format	51
4.2 Parameters in the Operand Field	53
4.3 Parameter Rules	53

4.4	General JCL Rules	54
4.5	Continuing JCL Statements	55
4.6	Commenting JCL	56
4.7	Style in Writing JCL	57
4.8	Placement of JCL Statements	58
<b>Chapter 5 The JOB Statement</b>		<b>60</b>
5.1	Jobname: Name of Job	62
5.2	Accounting Information	63
5.3	Name: Programmer Name	64
5.4	CLASS: Job Class	64
5.5	TIME: Time Limit	65
5.6	MSGCLASS: System Messages	67
5.7	MSGLEVEL: Printing JCL Statements	72
5.8	TYPRUN: Special Job Processing	73
	<i>Exercises</i>	74
<b>Chapter 6 The EXEC Statement</b>		<b>77</b>
6.1	Stepname: Name of Job Step	78
6.2	PGM: Name of Program	78
6.3	Procedure: Name of Cataloged Procedure	82
6.4	Keyword Parameters	83
6.5	Region Size	83
6.6	COND: Conditions for Bypassing Job Steps	85
6.7	IF/THEN/ELSE/ENDIF Statement Construct	92
6.8	PARM: Pass Parameters to Job Steps	97
6.9	ACCT: Job Step Accounting Information	99
6.10	SYSUDUMP, SYSABEND, SYSMDUMP: Abnormal Termination Dumps	100
	<i>Exercises</i>	101
<b>Chapter 7 The DD Statement</b>		<b>103</b>
7.1	Overview of Data Sets	103
7.2	Data Control Block	105
7.3	DD Statement Format	106
7.4	ddname: Data Definition Name	109
7.5	Referback: Referback Parameter	109
7.6	DCB: Data Control Block Parameter	110
7.7	DSN: Data Set Name	120
7.8	DISP: Data Set Disposition	125
7.9	UNIT: I/O Unit	135

7.10	VOL: Volume Parameter	140
	<i>Exercises</i>	145
<b>Chapter 8 More on the DD Statement</b>		<b>146</b>
8.1	Sequential and Partitioned Data Sets	146
8.2	DUMMY, NULLFILE: Dummy Data Sets	147
8.3	Concatenating Data Sets	149
	<i>Exercises</i>	152
<b>Chapter 9 DD Statements for Input Stream and Print Data Sets</b>		<b>153</b>
9.1	*,DATA: Input Stream Data Sets	153
9.2	SYSOUT: Output Stream Data Sets	155
9.3	The OUTPUT JCL Statement and Output DD Parameter	158
9.4	The JES /*OUTPUT Statement	168
9.5	The JES3 /*FORMAT PR Statement	169
9.6	Parameters Coded on Several Statements	170
<b>Chapter 10 Direct-Access Storage Devices</b>		<b>181</b>
10.1	Direct-Access Hardware Devices	181
10.2	Space Allocation	183
10.3	The SPACE Parameter	185
10.4	DCB Parameters	194
10.5	Virtual I/O (VIO) Temporary Data Sets	195
10.6	Estimating Space	196
10.7	The LABEL Parameter: Data Set Labels	200
10.8	Multivolume Data Sets	200
10.9	Using Data Sets on Direct-Access Volumes	201
10.10	ABSTR: Requesting Specific Tracks	205
	<i>Exercises</i>	206
<b>Chapter 11 SMS: Storage Management Subsystem</b>		<b>208</b>
11.1	The AVGREC Parameter	209
11.2	The DATACLAS, STORCLAS, MGMTCLAS, and SECMODEL Parameters	209
11.3	The LIKE and REFDD Parameters	213
11.4	RECORD and KEYOFF for VSAM Data Sets	214
11.5	DSNTYPE Parameter for Partitioned and Extended Sequential Data Sets	215

<b>Chapter 12</b>	<b><i>Magnetic Tapes</i></b>	<b>217</b>
12.1	Description of Tape	217
12.2	LABEL: Tape Labels	220
12.3	DCB Subparameters	228
12.4	Using Tapes	228
12.5	Compressing Data on Tape	231
12.6	Multivolume Tape Data Sets	232
12.7	Reading Tapes from Another Installation	233
12.8	ISO/ANSI/FIPS Version 3 Labels	233
	<i>Exercises</i>	235
<b>Chapter 13</b>	<b><i>JES2 and JES3</i></b>	<b>236</b>
13.1	Job Entry Subsystems	236
13.2	JES2	236
13.3	JES3	239
<b>Chapter 14</b>	<b><i>Cataloged and Instream Procedures</i></b>	<b>248</b>
14.1	Modifying Statements in Cataloged Procedures	249
14.2	Cataloged Procedures	255
14.3	Instream Procedures	257
14.4	Symbolic Parameters	258
14.5	Nesting Procedures and the INCLUDE Statement	267
14.6	DDNAME: Postponing Definition of Data Sets	270
14.7	Example of Cataloged Procedure	272
	<i>Exercises</i>	274
<b>Chapter 15</b>	<b><i>Generation Data Groups</i></b>	<b>276</b>
15.1	Creating the Generation Data Group Base Entry	276
15.2	Creating the Model Data Set Label (Non-SMS-Managed Data Sets Only)	277
15.3	Creating a Generation Data Set	278
15.4	Retrieving Generation Data Sets	280
15.5	Listing Generation Data Group Catalog Information	280
15.6	Deleting Generation Data Groups	280
<b>Chapter 16</b>	<b><i>Miscellaneous JCL Features</i></b>	<b>282</b>
16.1	Checkpoint/Restart	282
16.2	Spanned Records	289
16.3	Data Set Protection	289
16.4	Job Execution Priority	292
16.5	Other JCL Parameters	293



16.6	Null Statement	296
16.7	Operator Commands	297
16.8	CNTL/ENDCNTL Program Control Statements	298
16.9	XMIT Data Transmission Statements	298
<b>Chapter 17</b>	<b><i>VSAM Data Sets</i></b>	<b>300</b>
17.1	Creating VSAM Data Sets with JCL	304
17.2	Accessing VSAM Data Sets through JCL	305
17.3	The IDCAMS Utility	307
17.4	JOBCAT and STEPCAT DD Statements	314
<b>Chapter 18</b>	<b><i>The Linkage Editor and Loader</i></b>	<b>315</b>
18.1	The Linkage Editor	315
18.2	The Loader	328
	<i>Exercises</i>	329
<b>Chapter 19</b>	<b><i>IBM Utility Programs</i></b>	<b>331</b>
19.1	The IDCAMS Utility	331
19.2	The Ictool Utility Programs	336
19.3	The IBM Utility Programs	341
<b>Chapter 20</b>	<b><i>Sort/Merge</i></b>	<b>352</b>
20.1	Sorting Concepts	352
20.2	The DFSORT Program	353
20.3	The SORT Statement	354
20.4	MERGE Statement	358
20.5	Other SORT Statements	359
20.6	Sort Efficiency	376
	<i>Exercises</i>	378
<b>Chapter 21</b>	<b><i>ISPF</i></b>	<b>379</b>
21.1	Using ISPF	379
21.2	Using ISPF for Programming	384
21.3	Editing Data Sets	388
21.4	Browsing Text	408
21.5	The ISPF Utilities	409
<b>Chapter 22</b>	<b><i>TSO/E</i></b>	<b>417</b>
22.1	The TSO/E Language	417
22.2	Logging on and off TSO/E	419

22.3	Displaying Information about Data Sets	420
22.4	Allocating Data Sets	422
22.5	Calling Programs	424
22.6	Submitting Jobs	424
22.7	Use of TSO/E for Utility Functions	427
22.8	TSO/E CLISTS	428
 <b>Chapter 23 TSO/E REXX</b>		<b>438</b>
23.1	Variables	439
23.2	Arithmetic Expressions	440
23.3	Logical Expressions	440
23.4	Character Operations	441
23.5	REXX Statements	442
23.6	Supplying Arguments in the Command Line	447
 <b>Chapter 24 Hierarchical File System (HFS) Files</b>		<b>449</b>
24.1	JCL Parameters for HFS Files	449
24.2	TSO/E HFS Parameters	453
24.3	The BPXBATCH Utility	453
 <b>Chapter 25 JCL and the Internet</b>		<b>456</b>
25.1	Useful Web Sites	456
25.2	Sending E-Mail from Batch Jobs	457
 <b>Index</b>		<b>461</b>

---

## ***PREFACE***

The IBM mainframe continues to be alive and well, despite all the attention received by PCs. It has adapted to the changing environment, where issues such as security and reliability make it an ideal platform for the electronic economy. JCL (Job Control Language) is necessary to run batch jobs on IBM large mainframe computers, and batch jobs are an inherent part of programming on the mainframes. z/OS is the next generation of the OS/390 operating system. The information for it in this book has been extracted from close to 40 IBM manuals. There are so many manuals for z/OS that if programmers were each to have a personal copy of all the manuals they might need, they would risk crushing the continental plate under their weight. Since manuals are expensive, this book saves you and your company money.

This is the fifth edition of the JCL book. I have updated it to incorporate recent changes to JCL and the z/OS features.

My goal for this book continues to be to explain the operating system and provide readers with most of the information they need to use it. For this reason, chapters on such subjects as the linkage editor, the IBM utilities, the Sort/Merge utility, VSAM, TSO/E, ISPF, and REXX are included.

Please visit the book's Web site at [www.wiley.com/compbooks/brown](http://www.wiley.com/compbooks/brown) for solutions to the exercises and a summary of obsolete JCL features.

Gary DeWard Brown  
Los Angeles, California  
April 2002



---

## ***JOB CONTROL LANGUAGE PARAMETERS***

<b>Parameter</b>	<b>Subparam- eter of</b>	<b>Page</b>	<b>Parameter</b>	<b>Subparam- eter of</b>	<b>Page</b>
ACCODE	DD	234	COMPACT	OUTPUT	178
ACCT	EXEC	101	COMSETUP	OUTPUT	167
ADDRESS	OUTPUT	168	COND	JOB, EXEC	87
ADDRSPC	JOB, EXEC	293	CONTROL	OUTPUT	178
AFF	UNIT	140	COPIES	DD, OUTPUT	171
AMP	DD	303	CROPS	AMP	304
AVGREC	DD	209	DATA	DD	155
BLKSIZE	DD, DCB	118	DATAACK	OUTPUT	167
BLKSZLIM	DD	TK	DATACLAS	DD	209
BUFND	AMP	304	DCB	DD	112
BUFNI	AMP	304	DD	—	105
BUFNO	DD, DCB	120	DDNAME	DD	270
BUFSP	AMP	304	DEFAULT	OUTPUT	161
BUILDING	OUTPUT	168	DEN	DD, DCB	228
BURST	DD, OUTPUT	175	DEPT	OUTPUT	168
BYTES	JOB	173	DEST	DD, OUTPUT	173
CCSID	JOB, EXEC	TK	DISP	DD	127
CHARS	DD, OUTPUT	176	DLM	DD	156
CHKPT	DD	283	DPAGELBL	OUTPUT	167
CKPTLINE	OUTPUT	174	DSN	DD	122
CKPTPAGE	OUTPUT	174	DSNTYPE	DD	215
CKPTSEC	OUTPUT	174	DSORG	DD, DCB	113
CLASS	JOB	66	DUMMY	DD	149
CLASS	OUTPUT	166	DUPLEX	OUTPUT	TK
CNTL	—	296	DYNAMNBR	EXEC	294
CNTL	DD	296	ENDCNTL	—	296
COLORMAP	OUTPUT	TK	EXEC	—	79
COMMAND	—	295	EXPDT	DD, LABEL	289

Parameter	Subparameter of	Page	Parameter	Subparameter of	Page
FCB	DD, OUTPUT	175	OUTPUT	—	160
FILEDATA	DD	440	OUTPUT	DD	160
FLASH	DD, OUTPUT	177	OVERLAY	OUTPUT	TK
FORMDEF	OUTPUT	167	OVFL	OUTPUT	179
FORMLEN	OUTPUT	TK	PAGEDEF	OUTPUT	167
FORMS	OUTPUT	175	PAGES	JOB	173
FREE	DD	160	PARM	EXEC	99
GROUP	JOB	291	PASSWORD	JOB	291
GROUPID	OUTPUT	167	PATH	DD	439
HOLD	DD	158	PATHDISP	DD	440
IF/THEN/ ELSE/ENDIF	—	94	PATHMODE	DD	442
INCLUDE	DD	268	PATHOPTS	DD	440
INDEX	OUTPUT	179	PEND	—	257
JCLLIB	—	257	PERFORM	JOB, EXEC	293
JESDS	OUTPUT	166	PGM	EXEC	80
JOB	—	62	PIMSG	OUTPUT	167
JOBCAT	Special DD	311	PRMODE	OUTPUT	167
JOBLIB	Special DD	81	PROC	—	255
KEYOFF	DD	302	PROC	EXEC	85
LABEL	DD	220	PROTECT	DD	291
LGSTREAM	DD	TK	PRTERORR	OUTPUT	TK
LIKE	DD	213	PRTNO	OUTPUT	TK
LINDEX	OUTPUT	179	PRTOPTNS	OUTPUT	TK
LINECT	OUTPUT	178	PRTQUEUE	OUTPUT	TK
LINES	JOB	172	PRTSP	DD, DCB	178
LRECL	DD, DCB	116	PRTY	JOB	292
MEMLIMIT	JOB, EXEC	TK	PRTY	OUTPUT	167
MGMTCLAS	DD	211	QNAME	DD	294
MODIFY	DD, OUTPUT	176	RD	JOB, EXEC	284
MSGCLASS	JOB	69	RECFM	DD, DCB	114
MSGLEVEL	JOB	74	RECORG	DD	302
NAME	OUTPUT	168	REF	VOL	143
NOTIFY	JOB	415	REFDD	DD	213
NOTIFY	OUTPUT	166	REGION	JOB, EXEC	85
NULLFILE	DSN	149	RESFMT	OUTPUT	TK
OFFSET	OUTPUT	TK	RESTART	JOB	285
OPTCD	DD, DCB, AMP	176	RETAIN	OUTPUT	TK
OUTBIN	OUTPUT	166	RETRY	OUTPUT	TK
OUTDISP	OUTPUT	166	RETPD	DD, LABEL	289
OUTLIM	DD	172	RLS	DD	304
			ROOM	OUTPUT	168

<b>Parameter</b>	<b>Subparameter of</b>	<b>Page</b>	<b>Parameter</b>	<b>Subparameter of</b>	<b>Page</b>
SCHENV	JOB	TK	SYSOUT	DD	157
SECLABEL	JOB	291	SYSUDUMP	Special DD	102
SECMODEL	DD	212	TERM	DD	417
SEGMENT	DD	159	THRESHLD	OUTPUT	179
SER	VOL	143	TIME	JOB, EXEC	67
SET	—	263	TITLE	OUTPUT	168
SORTCKPT	Special DD	283	TRC	OUTPUT	176
SPIN	DD	159	TRTCH	DD, DCB	105
SPACE	DD	185	TYPRUN	JOB	75
STEPCAT	Special DD	311	UNIT	DD	137
STEPLIB	Special DD	82	USER	JOB	291
STORCLAS	DD	209	USERDATA	OUTPUT	168
STRNO	AMP	304	USERLIB	OUTPUT	168
SUBSYS	DD	295	VIO	DD	195
SYNAD	AMP	304	VOL	DD	142
SYMNames	DD	TK	WRITER	OUTPUT	174
SYSABEND	Special DD	102	XMIT	—	297
SYSAREA	OUTPUT	168	//*	—	58
SYSCHK	Special DD	287	//	—	295
SYSCKEOV	Special DD	283	/*	—	155
SYSIN	Special DD	155	*	DD	155
SYSMDUMP	Special DD	103			





---

## CHAPTER 1

---

# INTRODUCTION

### 1.1 THE SHOCK OF JCL

Your first use of JCL (*Job Control Language*) will be a shock. No doubt you have used personal computers costing \$500 or \$1,000 that had wonderfully human-engineered software, giving you an expectation of how easy it is to use a computer. Now, as you use a computer costing several million dollars, you may feel like a waif in a Dickens story standing in the shadow of a massive mainframe computer saying meekly, “Please, sir, may I run my job?” It will come as a shock that its software is not wonderfully human engineered.

The hardware and software design of large IBM mainframe computers date back to the days when Kennedy was president. JCL is a language that may be older than you are. It was designed at a time when user-friendliness was not even a gleam in the eye of its designers. This is easily demonstrated by taking the simple task of copying a file and contrasting how it is done through JCL with how it is done on the most popular personal computer system, Windows. To copy a file with Windows, you left-click twice on the MY COMPUTER icon, left-click on the C: drive icon, left-click twice on the folder containing the file, and right-click on the file to copy. On the resulting menu, you click on COPY and then left-click twice on the folder into which you want the file copied. Finally, you right-click inside the folder and select PASTE from the menu and you are done. It is wonderfully intuitive and simple. To do the same thing with JCL, you might write the following, in which *old-name* names the original file and *new-name* is the name you select for the copy.

```
//RT452216 JOB (45992,335),'SAMPLE JOB',CLASS=A
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD DSN=old-name,DISP=SHR
//SYSUT2 DD DSN=new-name,DISP=(NEW,CATLG),
```

```
//          UNIT=SYSDA,RECFM=FB,LRECL=80,
//          SPACE=(0,(100,20))
//SYSIN    DD *
          REPRO INFILE(SYSUT1) OUTFILE(SYSUT2)
/*
```

This is far more complicated and not remotely intuitive. But what if you must copy 50 files? With JCL, you could use your text editor to quickly make 50 copies of the JCL statements, and overtype all the *old-names* and *new-names*. Then you simply submit the job to the computer, which is easier than going through the Windows clicking and pointing procedure 50 times. Now suppose you must copy the same 50 files on a daily basis. Windows would be a nightmare, but with JCL, a single command lets you resubmit the lines of JCL that copy the 50 files.

Then the inevitable happens. Your boss comes in and, with an accusing eye, asks if you inadvertently forgot to copy a file the previous day. With the pointing and clicking of Windows, you would have no answer. By contrast, JCL produces a listing—an audit trail—that you can wave in your boss’s face to prove that the problem was not yours. The next day your boss informs you that the copy procedure must be done during the graveyard shift. With Windows, your nights will be sleepless. But with JCL, you can specify a job class that starts during the graveyard shift, leaving your nights free for more enjoyable activities.

What has just been described for JCL is the essence of *batch processing* (submitting one or many jobs or procedures in one fell swoop) and *production computing*. Interactive computing, such as that provided by Windows, Linux, Macintosh, and UNIX is wonderful for many things, but once those wonderful things are completed and the daily grind of running them begins, JCL comes into its own.

JCL is neither lovable nor simple. It was designed long ago and shows its age in ways such as having to code information in specific columns of a line. However, IBM has made changes to both JCL and the operating system that eliminate some of its worst aspects.

The large mainframe computer is an extremely conservative environment. The basic hardware architecture and operating system appear to the user much as they did over a third of a century ago. The benefit of this is that programs written back then can still run unchanged today, and knowledge gained back then is still valid. The greatest strength of large IBM mainframe computers, indeed, the strongest force in the computing universe, is *compatibility*. Billions of dollars are tied up in application software, and many companies would not accept an incompatible computer with modern design and software features even if it were free, because software is the dominating cost in computing, not hardware.

## 1.2 THE ROLE OF JCL

You do not use JCL to write computer programs. Instead, it consists of *control statements* that introduce a computer job to the operating system, provide accounting information, direct the operating system on what is to be done, request hardware devices, and execute the job. JCL tells the operating system everything it needs to know about a job's input/output (I/O) requirements. Sitting above JCL in many installations is a *job entry system (JES)* with a *job entry control language (JECL)*. You code the JECL statements to specify on which network computer to run the job, when to run the job, and where to send the resulting output. IBM provides two job entry systems for z/OS: JES2 for decentralized control, and JES3 for highly centralized control of several computers. JCL and JES go hand in hand, and this book describes both.

## 1.3 THE DIFFICULTY OF JCL

JCL provides the means of communicating between an application program and the operating system and computer hardware. Measured by the number of moving parts, the z/OS operating system is one of humankind's most complex creations. The computer hardware is less complex, but complex nonetheless.

JCL is difficult because of the way it is used. A normal programming language, however difficult, soon becomes familiar through constant usage. In contrast, JCL has language features used so infrequently that many never become familiar. JCL is also difficult because of its design. It is not a procedural language like COBOL or C/C++ in which you build up complex applications step by step from simple statements. JCL consists of individual parameters, each of which has an effect that may take pages to describe. JCL makes few useful assumptions for you—you must tell it exactly what to do. For example, virtually every batch computer program prints some output. However, the system doesn't assume this. You must supply a JCL statement to print output.

The z/OS operating system demands an extraordinary amount of information, most of it supplied by JCL, to run a job. For example, to save a file on disk storage, the system wants to know the record type, the record length, the block size, the type of I/O unit, the volume serial number of the disk volume, and more. (The recent changes to simplify JCL have been in this area.)

JCL was designed at a time when people were relatively cheap and computers expensive. In 1965, one million computer instructions were roughly

equivalent in cost to 15 minutes of a programmer's time. Today the opposite is true. People are expensive and computers are cheap. A million instructions today buys less than a fraction of a millisecond of a programmer's time. Consequently, JCL, designed to be efficient in computer time, is operating today in an environment where the cost of a person's time is the dominant expense. This book attempts to save *you* time—not just save *computer* time. Some of the worse things done in computing have been for efficiency. Saving two bytes in the year field is an extreme example that made the new millennium stressful for many. The cost performance of computer chips is still doubling roughly every 18 months, and while this can't last forever, it would be foolhardy to predict when it will end. This has a profound effect on how much effort a company wants to invest in optimizing programs.

## 1.4 THE APPROACH TO JCL

The first several chapters of this book describe the individual language statements, tell how to write them, explain what they do, and suggest how to use them. With this as background, the book shifts to functional descriptions of the hardware devices, access methods, and other topics. Because you rarely need many JCL features, the book notes whether a feature is **ESSENTIAL**, **SOMETIMES USED**, or **RARELY USED**. You should pick and choose.

The goal of this book is to give you all the information you need to program on z/OS, except for the programming language you use. To accomplish this, the book goes far beyond JCL. The book introduces the concepts and facilities of the operating system from the application programmer's point of view. Several non-JCL operating system facilities are also described, including IBM utility programs, the Sort/Merge program, the linkage editor, VSAM (Virtual Sequential Access Method), TSO/E (Time Sharing Option), and REXX (REstructured eXtended eXecutor).

This book explains JCL and shows you how to use it, but it won't try to make you like it because JCL is not a likable language. Almost no one, even programmers with years of experience, can sit down and write JCL statements the way they could COBOL or C++ statements. Consequently, people who write JCL generally know what they want to do and then consult some existing JCL to use as a prototype to write the new JCL statements. The book gives many short examples to serve as prototypes with the assumption that short, concise examples are preferable to lengthy explanations.

The book gives special attention to the use of JCL with COBOL, FORTRAN, PL/I, C/C++, and Assembler Language. Where appropriate, the book describes their interfaces to JCL. The book is based on the z/OS ver-

sion of the operating system. Although IBM occasionally releases new versions of z/OS, the new releases seldom change existing JCL.

If you are just learning z/OS, you can use this book as an introduction to the operating system and JCL, skipping over the **RARELY USED**, being selective with the **SOMETIMES USED**, and concentrating on the **ESSENTIAL** features. The book presumes you have some familiarity with a higher-level language. If you are an experienced z/OS programmer, you can use this book to learn unfamiliar JCL features or to refresh your memory on seldom-used features. Finally, the book serves as a reference for all who program in z/OS.

For classroom usage, read Chapters 1 to 14 in sequence, working in the installation's particular requirements. Then select topics from the remaining chapters as needed. Exercises at the end of many chapters consist of short, simple problems that can be run on a computer. They are designed as much to teach you about your installation and the problems of actually running jobs as they are about JCL as a language.

JCL provides many abbreviations and alternate names for coding. For simplicity and to reduce errors, the book shows only one form, the shortest, except when the short form conveys no meaning (CANCEL rather than C) or the long form is very short (NO rather than N). The book displays JCL statements and parameters in capital letters, with italics denoting items for which you select values. Text comments about a JCL statement are set off in brackets beneath the statement:

```
//SYSUT1 DD DSN=new-name, DISP=NEW
```

[The *new-name* represents a name you choose for the file. Code uppercase characters exactly as shown.]

---

## CHAPTER 2

---

# INTRODUCTION TO JCL AND z/OS

### Essential

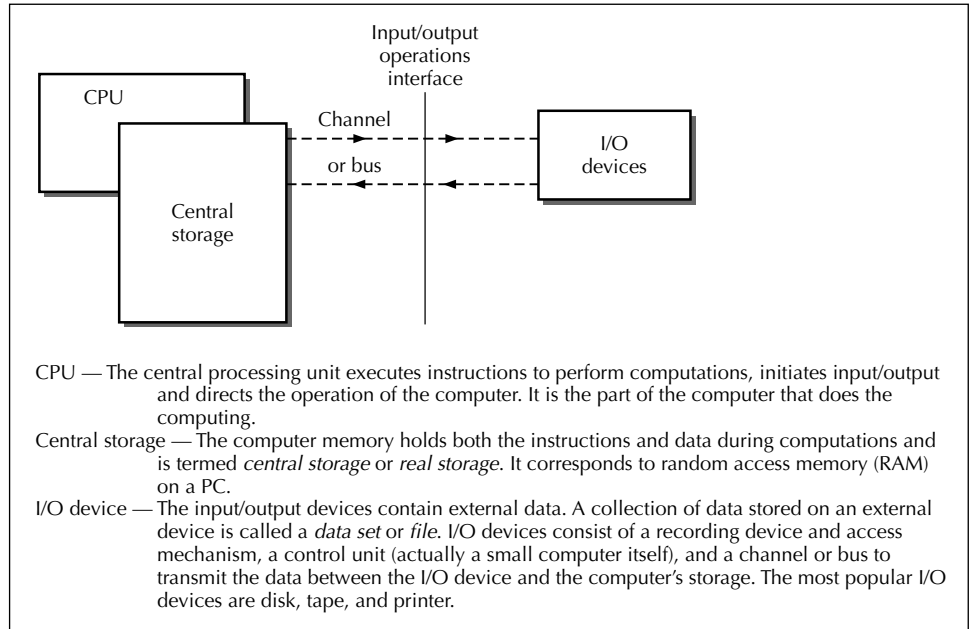
This chapter introduces z/OS concepts and vocabulary. You may be familiar with many of the concepts, but perhaps not in the z/OS context. While large mainframe computers have essentially the same hardware components as all other computers, including personal computers (PCs), the terminology differs. The formidable-sounding *direct-access storage device* (or *DASD*—pronounced *daz-dee*) on a mainframe becomes the more unassuming *hard disk* on a PC. Likewise, the impressive term *data set* on a mainframe turns out to be just a *file* on the PC.

The computer is essentially a device that reads some data, does some computing or processing, and writes some data. The main computer hardware that concerns you is illustrated in Figure 2.1 and consists of

- The CPU, or central processing unit. (Corresponding to a Pentium chip on a PC.)
- Central storage—formerly called real storage, computer memory, and core storage. (Corresponding to RAM, random access memory, on a PC.)
- Input/output (I/O) devices, often called peripherals, such as disks, tapes, and printers.

## 2.1 z/OS CONCEPTS AND VOCABULARY

The IBM Mainframe consists of a computer (the zSeries 900), an operating system (z/OS), and a vocabulary. People who work on z/OS form a separate culture, and you need to understand the concepts and speak the language to be accepted within the culture. Acronyms are the key to acceptance because everyone uses them, quickly forgetting what they represent.



**FIGURE 2.1** A typical computer.

### 2.1.1 The People

One difference between large IBM mainframes and smaller computers is the number of people required to support them. Computer operators run the mainframe, schedule jobs, mount tapes, and distribute output. One of the advantages of a large computer is that you don't have to worry about hardware problems or paper jams—the operators do this.

System programmers on large IBM mainframe computers install, maintain, and tune the operating system software. The software on large IBM mainframe computers is extremely complicated to install, and once installed, is even more difficult to maintain and tune for good performance. System programmers are highly skilled and valuable—and aware of this fact.

Installations usually have support staff to establish and enforce standards and procedures for using the computer. The support staff must also provide security, issue passwords and userids, and control access to the computer. Many installations also have a hot line or a help desk for answering questions. The first thing to do in any installation is to find out where to go for help.

### 2.1.2 The Operating System (z/OS)

The z/OS operating system introduces programs to the computer, initiates their execution, and schedules all the resources and services they require, such as printers, central storage, and disk storage space. The operating system is made up of a general library of programs that can be tailored to accommodate a variety of applications on a wide range of hardware configurations. The system programmers select the portions that an installation needs through a *system-generating process* (SYSGEN), add their own procedures, and update the procedures as the needs change.

The programs and routines that comprise the operating system are classified as either control or processing programs. *Control programs* perform the tasks of the operating system: reading, scheduling, initiating, allocating, executing, and terminating jobs in a continuous flow; supervising the dispatching and service requests of all work in the system; storing and retrieving all the data; controlling the user of virtual, central, and expanded storage; allocating the computer's resources; and finally, recovery from system and hardware failures.

*Processing programs* consist of *language translators* (such as the COBOL and C++ compilers), *service programs* (such as the linkage editor and sort programs), and *application programs* (such as the programs that you write.)

#### **Batch versus On-Line**

The original System/360 operating system was designed as a *batch system that could run on-line jobs*. This is in contrast to the PC, UNIX, and VM/ESA (the other popular system on large IBM mainframe computers), which were designed as *on-line systems that could run batch jobs*. With a *batch* system, you prepare a complete job and submit it to the computer. The computer's operating system schedules the job and executes it at its convenience, which may be hours later. You have no control over the job once you submit it. Often you submit a job and then hours later, when you get your output, find that a minor JCL error caused the entire job not to run. (With JCL, there is no such thing as a *minor* error.) *On-line* or *interactive* jobs are submitted from your computer terminal and you stay at the terminal while the job runs, usually interacting with the job to supply necessary information. You can quickly correct any errors.

Batch execution is great for long-running jobs, especially those run on a routine basis—*production* jobs. They have several advantages over interactive jobs:

You needn't wait at your terminal for your job to complete. You can go away and come back later.



You can run the job off shift, when computer rates are usually lower.

You can set up a complex job once and then keep resubmitting it rather than having to retype all the run commands each time.

The job may run more efficiently because it doesn't require the computer resources of an on-line job.

The computer can schedule the jobs at its convenience to make the most effective use of its hardware to maximize throughput.

Your program will produce an audit trail so you can reconstruct what happened during the run.

Of course, there are also disadvantages to running jobs in batch with JCL:

You don't get immediate turnaround, as a rule.

You must learn to use JCL.

If something goes wrong while a job is running, you don't get a chance to correct it and continue.

You must set the entire job up in advance, anticipating all the contingencies. You can't play it by ear.

You don't get to interact with the job and enter data and directions while it is running.

On-line is great for jobs requiring interaction while the job runs, jobs that change frequently, and jobs requiring quick turnaround. In practice, on-line is usually used for program development, data entry, inquiry, and quick response, whereas batch is used for production runs once the programs are developed.

Since many jobs are run concurrently on a mainframe computer, the resources of the computer must be shared so that each job eventually gets the resources it needs. This is done by multiprogramming and time-sharing.

### ***Time-Sharing and Multiprogramming***

*Time-sharing* allows many people to use a computer simultaneously in such a way that each is unaware that others are using the computer. The usual case is an on-line system with several consoles using the main computer at the same time. Time-sharing attempts to maximize an individual's use of the computer, not the efficiency of the computer itself. Time-sharing is supported on z/OS by such systems as the TSO/E (Time-Sharing Option), CICS (Customer Information Control System), IMS (Information Management System/Virtual Storage), VTAM (Virtual Telecommunications Access Method), and UNIX and Linux.

*Multiprogramming* is just the opposite of time-sharing in concept. It attempts to maximize the efficiency of the computer by keeping all the major components busy, such as the CPU, I/O devices, and central storage. Most jobs running on a large general-purpose computer do not use all the I/O devices or storage. Moreover, not all of the CPU is used since time is spent waiting for some I/O action to complete. Rewinding a tape is an extreme case in point.

Since most jobs do not use all of the storage, all of the I/O devices, or all of the CPU, a multiprogramming system can keep several jobs inside the computer at the same time and switch back and forth between them. Several jobs are loaded into storage and the operating system gives control to one job. It then switches control to another whenever one becomes idle. By balancing I/O-bound jobs with the compute-bound jobs, several jobs can be completed in little more time than it would take to complete a single job.

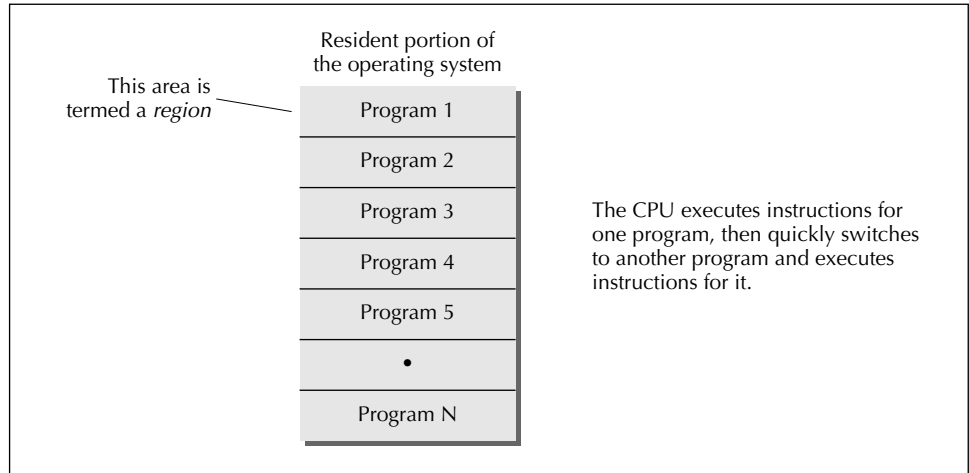
## 2.2 z/OS HARDWARE ARCHITECTURE

Perhaps the best way to understand the architecture is to trace its individual components to see why they were added. z/OS has evolved over the past third of century, and while not elegant or simple, it is eminently practical.

### 2.2.1 The 1960s and the Quest for Multiprogramming: MVT

Before the introduction of System/360 (z/OS's ancestor) in 1964, IBM computers ran a single program at a time. Business and scientific applications used different computers with different instruction sets and operating systems. Different-sized computers also had their own instruction sets and operating systems. System/360 gave all IBM mainframes the same hardware architecture so that applications could run independent of a particular computer model. The System/360 ancestor of today's operating system, MVT (Multiprogramming with Variable Tasks), could run a maximum of 15 jobs concurrently in the same central storage. The programs were placed in variable-sized areas called *regions* as shown in Figure 2.2. The computer used 24 bits for addressing, and so could address 16MB of memory or central storage. (The amount of storage that a computer can address is termed its *address space*, which is determined by the number of bits used to form an address. The address space size is calculated by the number of bits raised to the power of 2, so that 15 bits gives a  $2^{15} = 32,768$  address space.) However, memory was so expensive that a 512K machine was considered large.

Multiprogramming added immensely to the complexity of the operating system. The system had to protect each job in storage from other jobs. This was implemented through a hardware feature called *storage protection* in



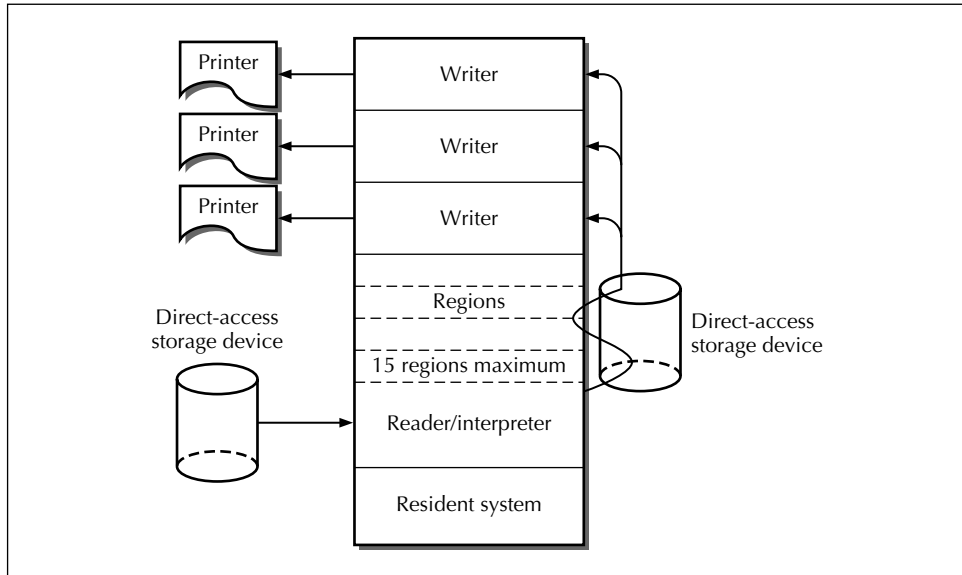
**FIGURE 2.2** Central storage in MVT.

which each region could change data only in its own region, protecting other regions and the operating system. The system also had to dole out the computer's resources so that the mixture of jobs in storage did not contend for nonshareable resources, such as tape drives. (PCs went through this barrier when Windows replaced MS-DOS.) The system had to hold back a job if it needed an unavailable resource and schedule another job whose resource requirements could be met.

The fact that the large mainframe computer was shared was a major reason it became so complex and difficult to use. Files and applications were potentially accessed by hundreds of users, making security a critical issue. If the mainframe crashed, hundreds of people could be affected, and in many cases, prolonged downtime could destroy a company. The mainframe, by necessity, became a serious environment.

The system, since it scheduled jobs based on their resource requirements, had to be told what resources each job needed. This was (and still is) done with JCL statements. For example, if a job needed a tape, a JCL statement had to describe the tape unit needed. The system would not schedule the job until such a tape unit became available. This prevented the job from sitting idle in central storage waiting for an available tape unit.

Figure 2.3 illustrates the original MVT multiprogramming system. Each job occupied a contiguous region in central storage, and the jobs remained there until they were complete. Users specified their region size with JCL statements. Some regions, such as the readers or writers, never completed and were always resident. The system decided which region to run and for how long and was made as crashproof as possible so that, although a particular job might fail, the system was not disturbed. This continues today,



**FIGURE 2.3** MVT multiprogramming system.

and much of the expense of the mainframe is in hardware and software designed to isolate problems and speed recovery. One of the advantages of z/OS as an old system is that most of the bugs have been shaken out and it is extremely reliable. (But not bugfree. Nothing as complex as an operating system is without bugs.)

The first region contained the *nucleus* or resident portion of the operating system (those portions of the system not kept on a direct-access storage device). The *reader/interpreter* (or *reader*) read in jobs and queued them on a direct-access volume. The term *volume* referred to a specific storage unit such as a direct-access storage volume or a tape cartridge. A direct-access storage volume was usually a disk pack—a large version of the hard disk on a PC. The *writers* wrote output from the queue on the direct-access volume onto the proper output device. Queuing the input and output on direct-access volumes was called *spooling*, which you might equate to the take-up reel of a movie projector. Spooling was an acronym for *simultaneous peripheral operation on-line*; one of those rare cases where an acronym conveys more meaning than that for which it stands. The sequence of lines read by the reader was called the *input stream*, and the sequence of output written by the writers was called the *output stream*.

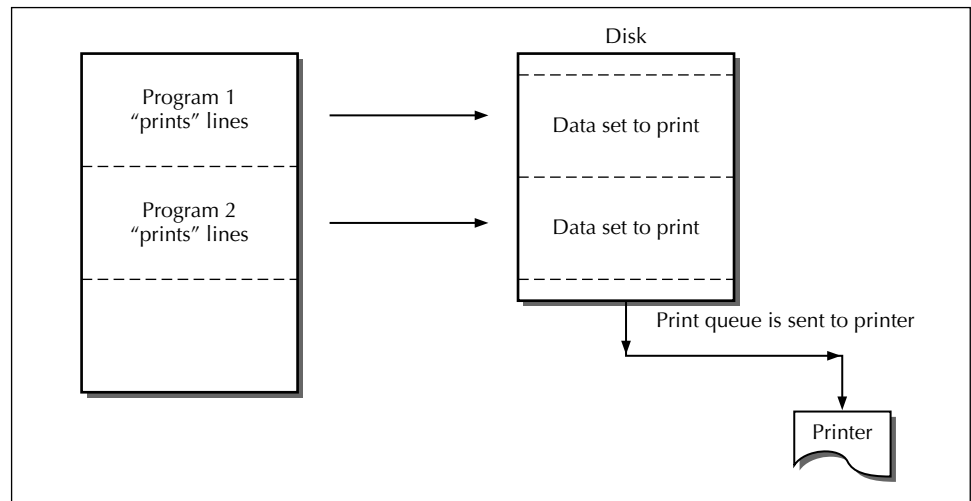
The *unit record* devices (printers, and, back then but never today, card readers and cardpunches) were normally assigned to the readers and writers. All other I/O units except tapes could be concurrently used by any of the regions. The system could assign a tape unit to only one region at a time.

This was one reason for using direct-access storage devices—the jobs were not kept waiting for a tape unit to become free.

To capsize the operating system: The reader/interpreter read jobs from the input stream and queued them on a direct-access volume. When a job came to the top of its queue, a system program called the *initiator/terminator* loaded it into a region and executed it. If the job read data from the input stream, the system fetched the lines from a direct-access volume where they had been stored. If the job printed output, this output was again queued on a direct-access volume. A job never printed directly—it was all done by queuing on a direct-access volume. The job was not aware of this because the system did it automatically. After the output was queued on a direct-access volume, the writers wrote it out to the appropriate output device. Figure 2.4 illustrates how spooling worked. All of this holds true today, except that users often examine their output on-line through a terminal while the output is in the output queue, and then decide whether to discard or print it. Although the running of a single region has been described here, several regions could be kept running concurrently in a similar manner.

Multiprogramming, which divided the computer's central storage into regions, presented several problems. A job was limited to the size of the largest region, and inevitably, some applications needed more central storage to run than the computer had. Central storage was often wasted when it became fragmented because jobs could only be run in contiguous storage.

Two applications aggravated these problems, time-sharing and teleprocessing. Time-sharing gave many people access to a computer at the same time with fast response required for each. *Teleprocessing* connected remote devices to the main computer through communication lines. The applica-



**FIGURE 2.4** Spooling.

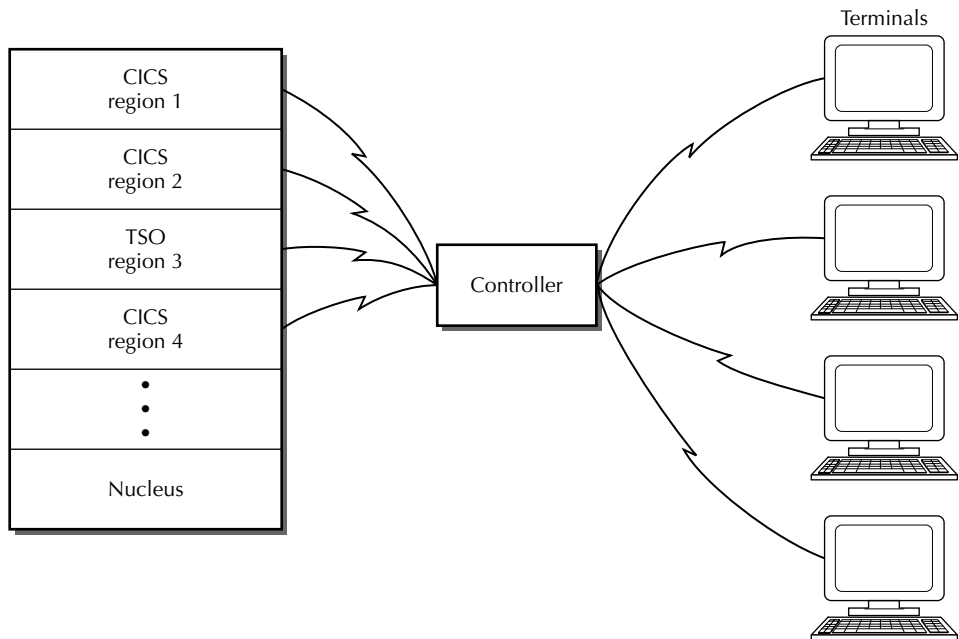
tions were often real time as typified by an airline reservations system in which an agent at a remote location reserved a seat on an airliner by communicating with the central computer. Time-sharing and teleprocessing applications were both characterized by long periods of inactivity and then brief spurts of frenzied activity requiring fast response.

Because the MVT was a batch system, on-line systems had to be developed that would run under its control in a region. Even today, such on-line systems as TSO/E, CICS, and VTAM are executed as batch jobs through JCL statements, and once running in their region, they begin supporting the many on-line users as illustrated in Figure 2.5.

### 2.2.2 The 1970s and the Quest for Central Storage: OS/VS2 and MVS/370

By the 1970s, the amount of central storage became a critical bottleneck. The operating system provided for 15 concurrent users, but there wasn't enough central storage for more than a few to reside in the computer at the same time. Central storage was still extremely expensive.

In 1972, IBM introduced the 370 family of computers with new circuitry that included a major architectural component called virtual storage. With *virtual storage*, the storage addresses of an application program were made independent of the addresses of the computer's central storage. The



**FIGURE 2.5** On-line system supporting many users.