IBM Data Processing Techniques

Core Requirements for 1130 FORTRAN

This manual provides information and techniques necessary to evaluate problems in terms of the approximate amount of 1130 object code required for legitimate FORTRAN source text. Initial discussion views the punched card and paper tape IBM 1130 Computing Systems. Later discussion is directed to the disk-oriented 1130.

Programming

CONTENTS

This publication is a minor revision of the earlier edition, C20-1641-0, which is obsolete. Revisions to the text are indicated with a vertical line to the left of the change.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

PUNCHED CARD AND PAPER TAPE SYSTEMS

The accompanying chart contains information about each subroutine that can be used in running 1130 FORTRAN object programs. An 1130 FORTRAN object program subroutine is utilized if the nature of the source text requires it.

Thus, for example, a program with the multiply symbol (*) surrounded by two floating-point operands will have the floating-point multiply subroutine in object core. In addition, this subroutine may require others for proper operation. A full discussion of these requirements is given here.

On the other hand, once the compiler determines that a requirement for an object time subroutine exists, the subroutine is loaded only once and may be used repeatedly from any point in the code generated from the source text.

Note that in the discussion that follows, certain separate subroutines use other subroutines in common. For instance both floating multiply and floating divide use a multiply routine of 66 words. These secondary routines should not be doubly counted. They appear only once in the object program even if all the primary subroutines that use them are employed.

Subroutine Name	Size in Words	Notes
1. Floating Add/Subtract — adds (subtracts) one floating-point number to (from) a second. X=A+B X=A-B	110	a,1
2. Floating Multiply — multiplies one floating-point number by a second. X=A*B	58	b,1
3. Floating Divide — divides one floating-point number by a second. X=A/B	90	c,1
4. Floating Subtract Reverse — reverses order of operands in a floating subtract if required (determined by the sequence of evaluation). X=A-B*C	26	d
5. Floating Divide Reverse — reverses order of operands in a floating divide if required (determined by the sequence of evaluation). X=(A+B)/(C+D)	30	e,m

Subroutine Name	Size in Words	Notes
6. Load/Store — moves data to and from an internal object time accumulator. Most frequently used subroutine. Should be included in every estimate. X=A	60	
7. Float/Fix — converts data from integer to real and vice versa. A=I I=A A=B + FLOAT (I) I=J + IFIX (B)	50	a
8. Reverse Sign — changes sign of object accumulator.	8	
9. Subscription — calculates relative object time addresses of subscripted variables. Used only when one or more subscripts are variables. Not used, for example, in case of A (2, 5, 9). Required in cases such as: A(I,J) A(2, 3, K) A(I, 2)	34	
10. Natural Logarithm B=ALOG(C)	150	b, d, e, f, m, n
11. Sine/Cosine A=SIN(B) A=COS(B)	130	d, f, m, n
12. Arctangent B=ATAN(C)	150	b, d, e, f, n
13. Square Root A=SQRT(B)	80	d, e, f, m, n
14. Exponential (e^X) A=EXP(B)	140	b, l, n
15. Exponentiation (A^B) A=B**C	60	f, g
16. Exponentiation (A^I) A=B**I	86	f, j
17. Exponentiation (I^J) I=J**K	76	
18. Absolute Value (Fl. Pt.) A=ABS(B)	10	
19. Absolute Value (Fx. Pt.) I=IABS(J)	16	

Subroutine Name	Size in Words	Notes
20. Card I/O — handles physical device.	56	h, k, o, p
21. Keyboard/Typewriter I/O — handles physical device.	80	h, k, o, p
22. Typewriter I/O	58	k, p
23. Paper Tape I/O — handles physical device.	216	k
24. Printer I/O — handles physical device.	200	
25. FORTRAN Format Interpreter — used to analyze format, convert decimal-binary-decimal, and control I/O subroutines. Called only if I/O operations are performed.	840	i
26. Subprogram Initializer — used to pass parameters to function subprograms, subroutine subprograms, and arithmetic statement functions.	32	
27. Hyperbolic Tangent A = TANH(X)	60	d, e, m, n, q
28. SIGN A = SIGN(B,C)	38	
29. ISIGN IA = ISIGN(IB,IC)	24	

NOTES

a. Requires normalizing routine of 42 words

b. Requires multiply routine of 66 words

c. Requires divide routine of 80 words

d. Requires subroutine 1

e. Requires subroutine 3

f. Requires subroutine 2

g. Requires subroutines 10, 14

h. Requires conversion routine of 60 words

i. Requires subroutines 6, 7

j. Requires subroutines 5, 6

k. Requires get-address routine of 16 words

l. Requires floating arithmetic range check routine of 34 words

m. Requires subroutine 6

n. Requires floating get parameter routine of 22 words

o. Requires Hollerith table of 54 words

p. Requires printer/EBCDIC table of 54 words

q. Requires subroutine 14

Object Time Core Analysis

The storage requirements for an object program involve four factors:

1. The code generated by the source text

2. The data storage required through the use of variable names, constants, and data declarations

3. The subroutines required to allow proper interaction of items 1 and 2

4. Overhead

Thus the FORTRAN statement

X = A + B

requires object code to add A to B and store the result in X (item 1), data cells for the variables X, A, and B (item 2), and the presence of subroutines 1 and 6 listed in the chart (item 3). In addition, certain overhead requirements (to be discussed) should be noted (item 4).

These four factors determine whether an object program is capable of residing in core.

Code Generated by Source Text

An accepted position regarding the average expansion factor from source text to 1130 object code is 14 words per FORTRAN source statement. While short statements (X=2., Y=A+B, GO TO 7, etc.) generate much less than this average, longer statements can generate more. While no definitive proof of this supposition exists, analogous expansion factors on a variety of other IBM equipment support this hypothesis. Thus a 100-statement, non-trivial source program may be said to generate 1400 words of 1130 object code.

Considering this problem as a retrofit, if data, subroutine, and overhead storage of an 8192-word 1130 require 2004 words of object core, a FORTRAN program consisting of 442 source statements ((8192-2004)/14) could conceivably be written.

Consequently, if data storage requirements for a job can be estimated, and the general functions required for execution of the job can be determined, a retrofit will then determine the number of FORTRAN statements that can be written before memory is exhausted.

While it may not be possible to estimate exactly how many FORTRAN source statements are necessary to produce a completed program, a retrofit, as described above, will give useful information. Thus, if it is determined that 2108 words are required for data and subroutine storage for a particular problem, the number of FORTRAN source statements that can be serviced in a 4096-word 1130 is about 142. If, in actuality, a a similar problem takes 400 to 500 FORTRAN statements, either a larger machine (core, disk, or both), or segmenting is required.

Data Storage Requirements

Floating-point data in the 1130 FORTRAN system is stored as either two words (normal precision) or three words (extended precision). Fixed-point data is stored as either one, two, or three words:

1. One-word storage. A configuration card at compile time specifies that integers are to occupy one word at object time.

2. Two-word storage. The default condition for normal precision is two-word storage for integers.

3. Three-word storage. All integers are automatically three words long if (a) floating-point extended precision is requested, and (b) one-word storage is not specifically requested.

Thus, a FORTRAN program consisting of a DIMENSION statement that specifies a 10 x 10 floating-point array, the occurrence of 25 other floating-point variables and constants, and ten fixed-point variables and constants occupies (normal precision) 260 words at object time if integers are stored as one-word units. This same number of variables and constants occupy 270 words if the integer default condition exists. (This example assumes no EQUIVALENCE statements.)

In the case of extended precision (floating-point) and one-word-integer storage, the previous example would occupy 385 words of object storage. Lastly, extended precision (floating-point) and three-word-integer storage would indicate an object time data storage requirement for 405 words. It is not possible to array floating-point in extended precision and at the same time request two-word-integer arithmetic. It is also not possible to have normal precision floating-point and three-word-integer arithmetic. Note that fixed-point arithmetic uses 16 bits regardless of the internal word size.

Subroutine Requirements

As implied earlier, object time subroutine requirements are a function of the source text. As an example, the FORTRAN statement

```
A = B + C * D/E -F
```

would require the object time presence of subroutines 1, 2, 3, and 6, and the ones they call, but no others.

The FORTRAN statement

```
A = B * * C
```

would require subroutines 15 and 6, but no others.

The FORTRAN statements

```
A = B + C
X = Y + Z
```

require subroutines 1 and 6 but no others. The second appearance of the "+" does not require a second appearance of subroutine 1.

The "usual" FORTRAN source program generates a requirement for:

1.	Floating add/subtract	(110 + 34 + 42)
2.	Floating multiply	(58 + 66)
3.	Floating divide	(90 + 80)
4.	Floating subtract reverse	(26)
5.	Floating divide reverse	(30)
6.	Load/store	(60)
7.	Reverse sign	(8)
	TOTAL	604 words

The figure should be a minimum starting point for basing estimations. Of course, if one never divides in a program, the core required would be 170 words less, but this is highly unlikely.

Overhead

All 1130 FORTRAN object programs require certain overhead core because of the design of the compiler. The following overhead must be included in object time size analysis:

Name	Function	Size in Words
Interrupts	Hardware words required to be serviced by I/O routines.	50
Loader loss 1	Core image loader is used if object program has been converted by IBM-supplied core image converter program before problem execution.	Maximum 40, but reduced for each variable (except common) that is stored there. If, for example, 20 two-word-integer variables are used in the source text, the loss is zero words.
Loader loss 2	Used if direct execution of object program is requested. Serves as both relocator and loader.	Maximum 450, but reduced for each variable (except common) which is stored there. Thus, a normal precision 15 x 15 floating-point array will result in a loss of zero words.
Transfer vectors	Linkage mechanism for sub-routines.	Variable with maximum of 117 words.
Buffering	Used in I/O. Always present.	121

Example

Consider a 4096-word 1130 card system. An estimate of function for a program is:

1. Reading/writing data from/to cards
2. Use of the console typewriter but not the keyboard
3. The "usual" floating-point arithmetic subroutines as discussed earlier
4. Arctangent

An estimate of data storage is:

1. Thirty variables and constants (floating-point, normal precision)
2. Ten variables and constants (fixed-point, one-word)
3. Two 8 x 8 arrays (floating-point, normal precision, non-EQUIVALENCEd)

An estimate of overhead is:

1.	Interrupt service		50 words
2.	Loader loss		0 words
3.	Transfer vectors	(≈)	69 words
4.	Buffering		121 words

Using the information above, the following conclusions are made:

1. Subroutine storage

a.	I/O (56 + 184 + 58 + 840 + 50)		1188
b.	Arithmetic		604
c.	Arctangent		172
d.	Subscription		34
		Total	1998 words

2. Data storage

a.	30 floating-point variables and constants		60
b.	10 fixed-point variables and constants		10
c.	Arrays		256
		Total	326 words

3. Overhead — Total 240 words

Grand Total 2564 words

4. Remaining for object code (4096-2564) — 1532 words

5. Retrofit (1532/14) — 109 FORTRAN statements

The analysis made up to this point concerning the card/paper tape FORTRAN system must be amplified for the disk FORTRAN system.

In order to process the largest possible program, the disk FORTRAN system will have one or two automatic overlay levels. When the disk is used for system residence, one overlay level will be possible. When the disk is used as a device to and from which data is written, in addition to its function for systems residence, two overlay levels will be possible. If an object program and its subroutines are too large to reside in core simultaneously, the subroutines will be core-disk overlaid in a certain fashion to allow for more object code. Should this fail, the program is said not to fit in the available core under the automatic segmentation scheme.

Such an overlay scheme has a positive effect on the amount of compilable source text. However, a performance degradation can occur for each overlay. Thus, if a large program must run with overlay, it could (but may not necessarily) run more slowly than if it had been able to fit with no overlay.

Program Size Implications

It is important to understand how the overlay scheme operates, so that the available amount of core for the object program can be determined.

DISK USED FOR SYSTEM RESIDENCE ONLY

If the disk is used only for system residence and not as a device to and from which data is transferred, the following changes should be noted in the object time storage requirements:

1. Interrupt servicing and buffer area are reduced to zero words.

2. If the object program is disk-resident, the loader loss is zero. It remains as previously described under card/paper tape FORTRAN if the object program is loaded into core from cards or paper tape.

3. The maximum number of transfer vectors is increased by 30 words.

4. The resident monitor for use with FORTRAN, which is not in the card and paper tape system, occupies 400 words at all times.

At the no-overlay level, the system operates in the fashion described for card and paper tape FORTRAN. The previous example thus becomes:

1. Subroutine storage (no change) 1998 words

2. Data storage (no change) 326 words

3. Overhead
   a. Loader loss 0 words
   b. Transfer vectors ($\approx$) 90 words
   c. Monitor 400 words

   Total 490 words

   Grand Total 2814 words

4. Remaining for object code (4096 - 2814) 1282 words

5. Retrofit (1282/14) 91 FORTRAN statements

At the overlay level, the arithmetic and functional subroutines constitute one package, and the I/O subroutines constitute a second. The larger of the two determines the subroutine storage. There are three subroutines potentially common to both the arithmetic and functional package and the I/O package. If they are required by either package they appear in neither but are removed to the overhead area. They are:

1. Fix/Float
2. Load/Store
3. Subscription

The previous example thus becomes:

1. Subroutine storage
   a. Arithmetic and functional subroutines: floating point (604) less load/store (60) plus arctangent (172) less subscription (34) = 682
   b. I/O subroutines (1188) less fix/float (50) = 1138

   Larger: 1138 words

2. Data storage (no change) 326 words

3. Overhead
   a. Transfer vectors and monitor 490 words
   b. Load/store 60 words
   c. Subscription 34 words
   d. Fix/float 50 words

   Total 634 words

   Grand total 2098 words

4. Remaining 1998 words

5. Retrofit 143 FORTRAN statements

DISK USED AS AN I/O DEVICE

If the disk is being used for the storage of data in addition to its function for system residence, one can achieve two levels of overlay. Changes should be noted in the object time storage requirements:

1. Same as 1, 2, 3, and 4 when disk is used for system residence only.

2. A disk overhead buffer of 322 words is required.

3. A disk format interpreter of 280 words is required in the overhead area.

At the no-overlay level, the system operates in the fashion described for card and paper tape FORTRAN. If the previous example has the additional requirement of reading and writing data on disk, the analysis becomes:

1.	Subroutine storage		
	a. Card I/O (56 + 184)		240 words
	b. Typewriter I/O		58 words
	c. Format interpreter (840 + 50)		890 words
	d. Arithmetic		604 words
	e. Arctangent		172 words
	f. Subscription		34 words
		Total	1940 words
2.	Data storage (no change)		326 words
3.	Overhead		
	a. Loader loss		0 words
	b. Transfer vectors	(≈)	90 words
	c. Buffering		322 words
	d. Monitor		400 words
	e. Disk format interpreter		280 words
		Total	1092 words
		Grand Total	3358 words
4.	Remaining		738 words
5.	Retrofit		52 FORTRAN statements

At the first overlay level the technique is identical with that of the overlay level where disk is not used for data transfer. Only the figures change. The previous example thus becomes:

1.	Subroutine storage	
	a. Arithmetic and functional	702 words
	b. I/O subroutines (56 + 184 + 58 + 840)	1138 words
	Larger:	1138 words
2.	Data storage (no change)	326 words
3.	Overhead — Loader loss, transfer vectors, buffer, monitor, load/store, subscription, disk format interpreter	1188 words
	Grand Total:	2652 words
4.	Remaining	1444 words
5.	Retrofit	103 FORTRAN statements

At the second overlay level, the arithmetic and functional subroutines (less the three mentioned) constitute one overlay. The non-disk I/O constitutes a second overlay. The disk I/O contitutes a third overlay. The non-disk I/O consists of an 840-word format interpreter plus those I/O routines that control the devices. The disk I/O consists of a constant 602 words including its buffer of 322 words, which is not in the overhead area for this overlay level, and the disk format interpreter. The previous example thus becomes:

1.	Subroutine storage	
	a. Arithmetic and functional	692
	b. Non-disk I/O (56 + 184 + 58 + 840)	1138
	c. Disk I/O	602
	Largest of the three:	1138 words
2.	Data storage (no change)	326 words
3.	Overhead (400 + 90 + 60 + 34 + 50)	634 words
	Grand Total:	2098 words
4.	Remaining	1998 words
5.	Retrofit	143 FORTRAN Statements

Program Speed Implications

If the overlay scheme is required, program load time will be increased in a typical case by about 14 seconds, with the maximum possible increase being about 37 seconds.

Such an overlay scheme will also produce a degradation of execution time relative to the execution time required without overlays employed. In the examples below, 1 is considered to be the execution time with no overlays required. For example, 1.5 as a performance ratio indicates that execution time will be one and one-half times that required without overlays.

In order to analyze this degradation, three machine configuration cases have been selected, each with a worst and best case problem program.

CASE 1

An individual has a card 1130 and upgrades to a card/file 1130 using Monitor system. As the program will now operate within the Disk Monitor, which utilizes the first 400 words of storage, thus providing less object core than available on a non-Monitor system, it is possible that an overlay may be necessary. Assume a configuration with a 1442 card reader, Model 7, which is the faster of the two available.

His source programs are of two categories:

1. Worst case: Every other statement is a card read or write statement.

2. Best case: The initial source program statements are card read. The final source program statements are card write. Between these read and write statements, only computation takes place.

If his program runs at the first overlay level, he may expect a performance of 1 (plus a fixed increment of about one second) for the best case and 1.5 for the worst case.

In normal usage, programs will fall between the worst and best cases described above. No case can produce a performance degradation worse than the defined worst case. If the 1442, Model 6, is employed, the throughput degradation will be smaller than that described above. This is because the most rapid I/O device, when delayed, causes the greatest percentage of throughput degradation.

CASE 2

This is the same as case 1, but the initial system is a paper tape 1130 and the upgraded system is paper tape/file 1130.

1. Worst case: Every other statement is a paper tape read or write statement.

2. Best case: The initial source program statements are paper tape read. The final source program statements are paper tape write. Between these read and write statements, only computation takes place.

The following tabulation describes the throughput degradation that may be expected:

	Worst Case	Best Case
No overlay	1	1
First level	1.1	1 (+1 second)

CASE 3

In this last case, we consider the effect of the overlay scheme for programs running without overlay on a file system, which are then enlarged (by additional source text) so that they do require overlay. Another way of stating this is to assume that every program does not require overlay. What is the performance degradation to be expected if overlay is then required? This information will aid one in deciding whether 4K or 8K disk is more applicable to a particular situation.

Again, it is necessary to consider a best and worst case:

1. Worst case: The source program has the following three statements repeated:
   Read cards
   Read file
   Process
   In addition, consider the number of cylinders to be covered between the overlaying subroutines and the data being read to be the most (160) or the least (2).

2. Best case: The source program has no disk I/O statements, but rather only card or paper tape ones. The input statements are first and the output statements are last. Further, the use of locals is not employed.

The following tabulation describes the throughput degradation that may be expected:

	Worst Case	Best Case
No overlay	1	1
First level	7.7 furthest arm motion	1(+1 second)
	1.6 least arm motion	1(+1 second)
Second level	7.9 furthest arm motion	
	1.7 least arm motion	

Printed in U.S.A. C20-1641-1

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York 10601