How to master Assembler Code?
Work on it as easy as on COBOL

ITP Software Systems Ltd.

Master your software with ITP-PANORAMA

www.itp-panorama.com

Introduction:
Many companies still have a significant part of their applications code in Assembler.

Assembler Programs are extreme problematic to maintain. (Never touch a running system)

Searches in high-level programming languages like Cobol or PL/I consume some 40% of the total time for a change in an application. This search time can be reduced by 90-95% using ITP-PANORAMA.

Searches in Assembler programs eat up nearly 80% of the money spent for any change.

Using ITP-PANORAMA the search time in Assembler programs can be reduce by 50%.

History:
Before 1980 Assembler was very common. Performance critical modules had to be coded in Assembler to make applications fast enough on the hardware of this time.

Other programming languages did not offer access to system functions.

Assembler did not have restrictions of higher level languages like dynamic linkage and the ability to modularize.

The Challenge:
The number of developers who are familiar with Assembler decreases rapidly. The new generation of developers had no opportunity to learn this language.

Even small changes can cost a lot of money which prevents changes from being made.

-Assembler programs are blocking considerations of migration. (new hardware, data base etc.)
-Various addressing modes (AMODE24, AMODE31 ...) cause dependencies on calls, linkage and so on.

ITP-PANORAMA Technology
Assembler programs can be scanned together with programs written in higher languages. This allows searches across the entire application system. The user interface shows Assembler very much like Cobol.
ITP-PANORAMA can handle Assembler specifics:

Assembler statements referring to a memory location are handled like statements in other languages together with the addressed symbol (location).

Symbol tables are built up the same way they are used by the Assembler compiler, with all attributes like start address, length and type.

The presentation, as far as possible, follows the rules of COBOL / PL/I record definitions with level number and re-definition information. This is based on a specific handling of DSECT's and ORG-statements.

Handling of Assembler Macros:
User macros are expanded so that statements and symbols that may contain generated statements and symbols are displayed. A reference is maintained to the original record in the macro. This can be tracked to understand which path in the macro produced the information.

Some system-macros are interpreted by the scanner (DCB, OPEN, GET, PUT, LOAD, ATTACH ...). They are presented like similar functions in high-level languages. File-access, calls between modules (often including a parameter list) are recognized.

The remaining macros are expanded. To achieve this the system-macro libraries (SYS1.MACLIB, IMSLIB etc) need to be loaded from the host.

Parameters passing thru registers often use constructs like:

```
LA      R1,Symbol
OPEN    (R1)
```

and similar sequences. A small register memory helps to present the right symbol as a reference for the macro. This memory will be cleared at all levels.

Hints for Re-Implementation:
Studying the Assembler part of large application systems lead us to the following guidance:

1. Programs that can be re-written using higher level languages
Data manipulation programs that are today written in a high-level language would show information on the environment and also the associated records (Files, IMS segments, DB2 table accesses)

How to master Assembler Code?
Work on it as easy as on COBOL

Using ITP-PANORAMA the presentation is quite similar to high-level languages. Therefore it is possible to rewrite the code almost immediately. Even computation between fields can be followed up if the programmer systematically used symbols for addressing.

2. Programs that use system functions
In programs that are heavily using system macros, ITP-PANORAMA shows which macros are used together with the parameter preparation. In this case it could be feasible to rewrite such a program in C, where most of the system functions exist in the form of run time functions.

Migrating to another platform can often eliminate the need for such programs (the same functionality can be achieved in a different way).

3. Nearly un-understandable programs
In the case of very chaotic programs that are mainly working with registers, it may help to study the high-level programs calling them and trying to understand the functions from the usage.

Such programs can be used to implement string manipulation functions missed in higher level languages. Understanding the functionality, they can be easily rewritten in C.

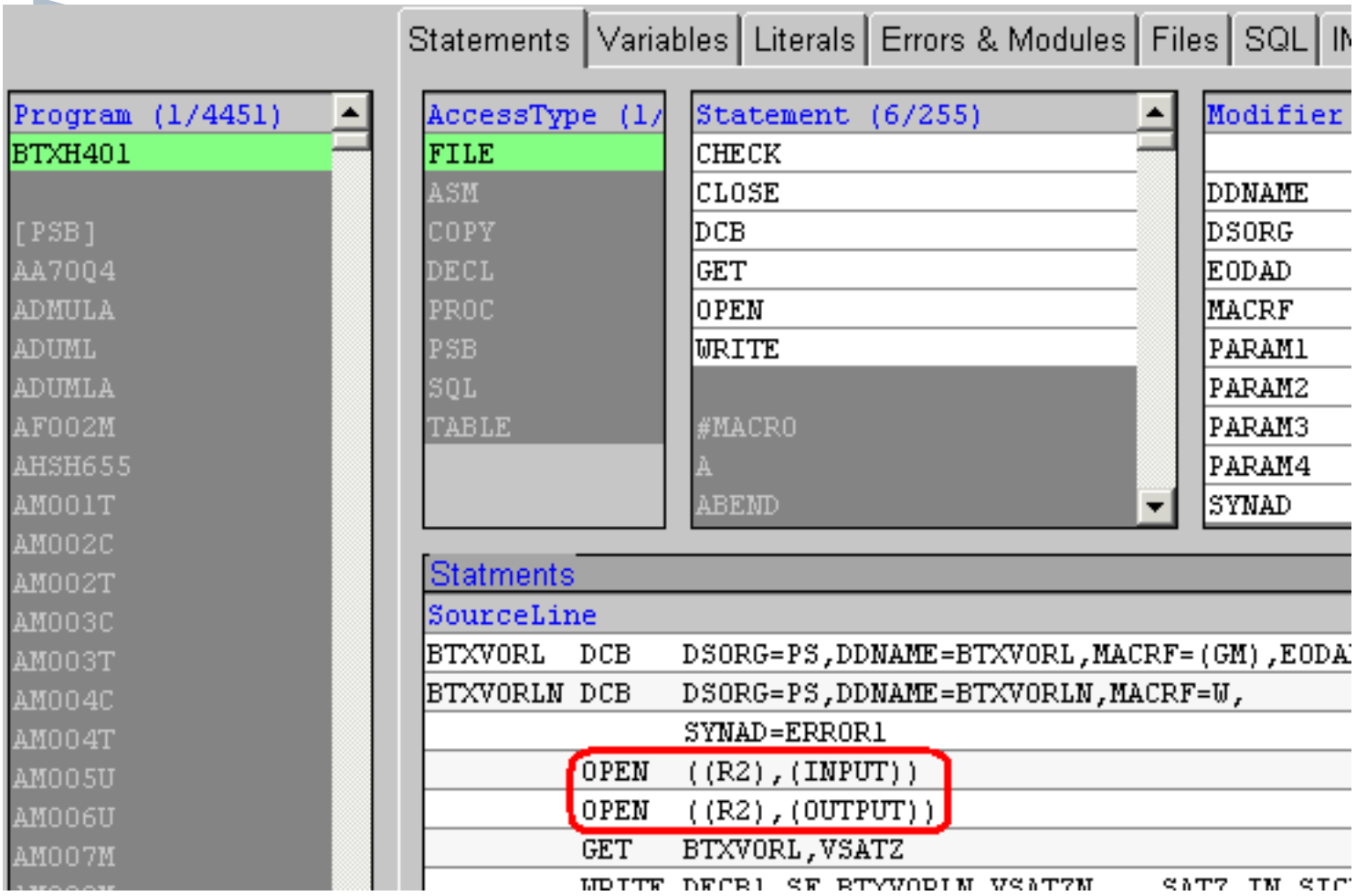
Advantages of using ITP-PANORAMA:
If programs fall under one of the above categories, it is possible to estimate the time that a reimplementation will take.

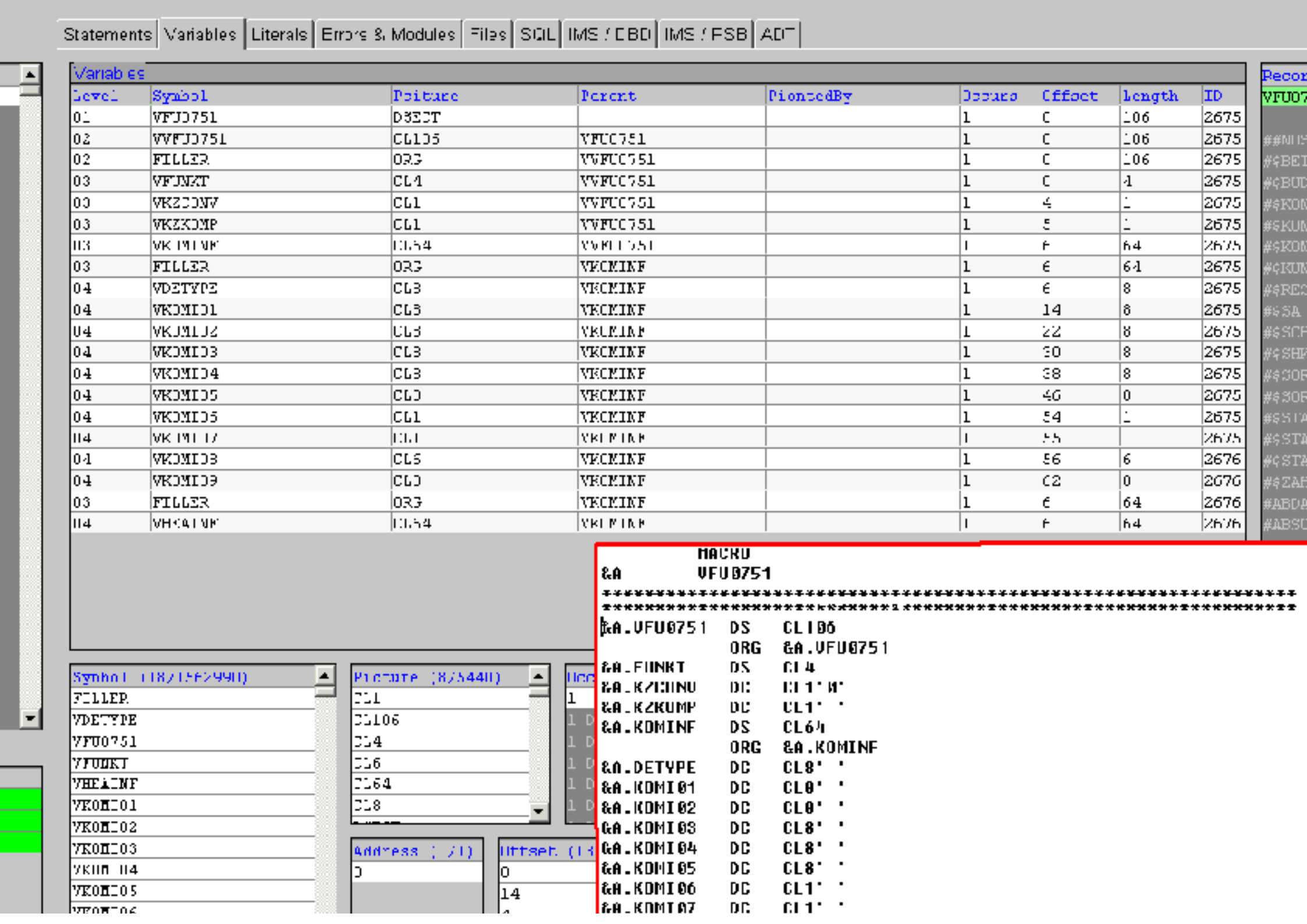
In many cases reimplementation can be done fairly mechanically by just following the displays of ITP-PANORAMA.

Even a JAVA developer can navigate thru an application written in Assembler because the user interface is presenting the code in an easy to understand fashion.

ITP-PANORAMA is an absolutely necessary tool if Assembler programs have to be examined and understood.

ITP-PANORAMA will take Assembler out of its isolation and let a developer see applications written in Assembler as part of the entire system.




Statements
Variables
Literals
Errors & Modules
Files
SQL
IMS / DBD
IMS / PSB
ADT
Variables
Level
Symbol
Picture
Parent
RedefinedBy
Occurs
Offset
Length
ID
01 VFU0751 DSECT 1 0 106 2675
02 VVFU0751 CL106 VFU0751 1 0 106 2675
02 FILLER ORG VVFU0751 1 0 106 2675
03 VFUNKT CL4 VVFU0751 1 0 4 2675
03 VKZKONV CL1 VVFU0751 1 4 1 2675
03 VKZKOMP CL1 VVFU0751 1 5 1 2675
03 VKOMINF CL64 VVFU0751 1 6 64 2675
03 FILLER ORG VKOMINF 1 6 64 2675
04 VDETYPE CL8 VKOMINF 1 6 8 2675
04 VKOMI01 CL8 VKOMINF 1 14 8 2675
04 VKOMI02 CL8 VKOMINF 1 22 8 2675
04 VKOMI03 CL8 VKOMINF 1 30 8 2675
04 VKOMI04 CL8 VKOMINF 1 38 8 2675
04 VKOMI05 CL8 VKOMINF 1 46 0 2675
04 VKOMI06 CL1 VKOMINF 1 54 1 2675
04 VKOMI07 CL1 VKOMINF 1 55 2675
04 VKOMI08 CL8 VKOMINF 1 56 6 2676
04 VKOMI09 CL8 VKOMINF 1 62 0 2676
03 FILLER ORG VKOMINF 1 6 64 2676
04 VHEADINF CL64 VKOMINF 1 6 64 2676
Symbol (18/1562/4480)
FILLER
VDETYPE
VFU0751
VFUNKT
VHEADINF
VKOMI01
VKOMI02
VKOMI03
VKOMI04
VKOMI05
Picture (8/5440)
CL1
CL106
CL4
CL6
CL64
CL8
Address (1/1)
0
Offset (14
0
14
MACRO
&A VFU0751
&A.VFU0751 DS CL106
ORG &A.VFU0751
&A.FUNKT DS CL4
&A.KZKONV DC CL1' '
&A.KZKOMP DC CL1' '
&A.KOMINF DS CL64
ORG &A.KOMINF
&A.DETYPE DC CL8' '
&A.KOMI01 DC CL8' '
&A.KOMI02 DC CL8' '
&A.KOMI03 DC CL8' '
&A.KOMI04 DC CL8' '
&A.KOMI05 DC CL8' '
&A.KOMI06 DC CL1' '
&A.KOMI07 DC CL1' '