Supporting the Specification and Analysis of Timing Constraints

by

Lo Ko, Christopher Healy, Emily Ratliff,
Robert Arnold, David Whalley
Florida State University

and

Marion Harmon
Florida A&M University

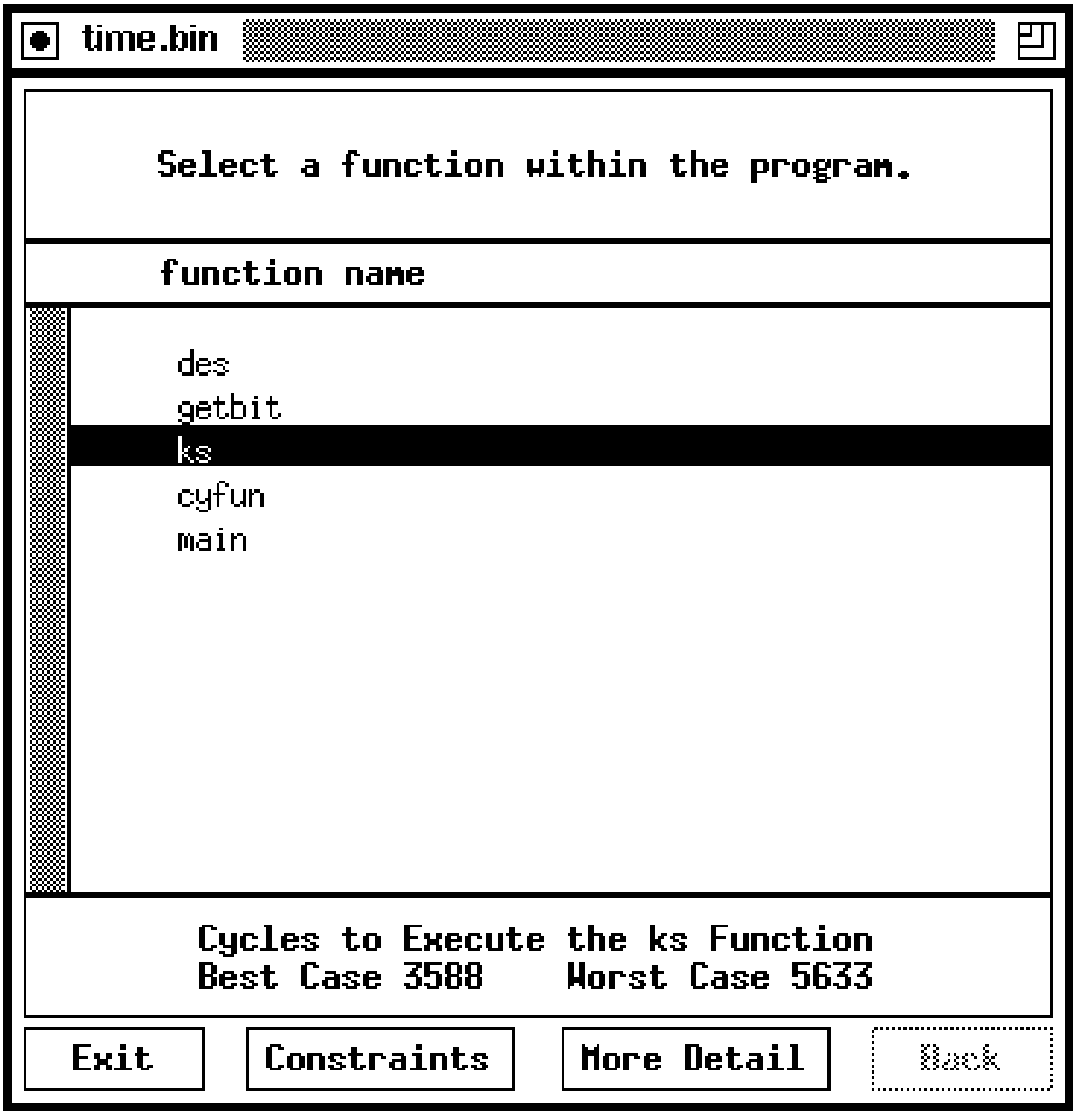
The Problem

- High-level timing analysis allows a user to relate timing constraints to source code portions.
- Low-level timing analysis on machine code is much more accurate.
- How can a user specify timing constraints at the source code level and obtain timing predictions associated with the more accurate low-level analysis?

Goals

- A user should be able to quickly specify constraints and obtain timing predictions for the specified portions of a program.
- The user should only be allowed to select portions of the program for which timing bounds can be obtained.
- The ability to specify constraints and obtain timing predictions should not inhibit compiler optimizations from being performed.
- The correspondence between source code and machine code of the program selected by the user for timing prediction should be graphically depicted.

Main Window at Function Level



Source Code Window

C Source Code of des.c

```
line # source code
 23          49,17,57,25};
 24       static great kns[17];
 25       static int initflag=1;
 26       int ii,i,j,k;
 27       unsigned long ic,shifter,getbit();
 28       immense itmp;
 29       void cyfun(), ks();
 30
 31       if (initflag) {
 32          initflag=0;
 33          bit[1]=shifter=1L;
 34          for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
 35          }
 36       if (*newkey) {
 37          *newkey=0;
 38          for(i=1;i<=16;i++) ks(key, i, &kns[i]);
 39          }
 40       itmp.r=itmp.l=0L;
 41       for (j=32,k=64;j>=1;j--,k--) {
 42          itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
 43          itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
 44          }
 45       for (i=1;i<=16;i++) {
 46          ii = (isw == 1 ? 17-i : i);
 47          cyfun(itmp.l, kns[ii], &ic);
 48          ic ^= itmp.r;
 49          itmp.r=itmp.l;
 50          itmp.l=ic;
 51          }
 52       ic=itmp.r;
 53       itmp.r=itmp.l;
 54       itmp.l=ic;
 55       (*out).r=(*out).l=0L;
 56       for (j=32,k=64; j >= 1; j--, k--) {
 57          (*out).r = ((*out).r <<= 1) | getbit(itmp,ipm[j],32);
 58          (*out).l = ((*out).l <<= 1) | getbit(itmp,ipm[k],32);
 59          }
 60    }
```

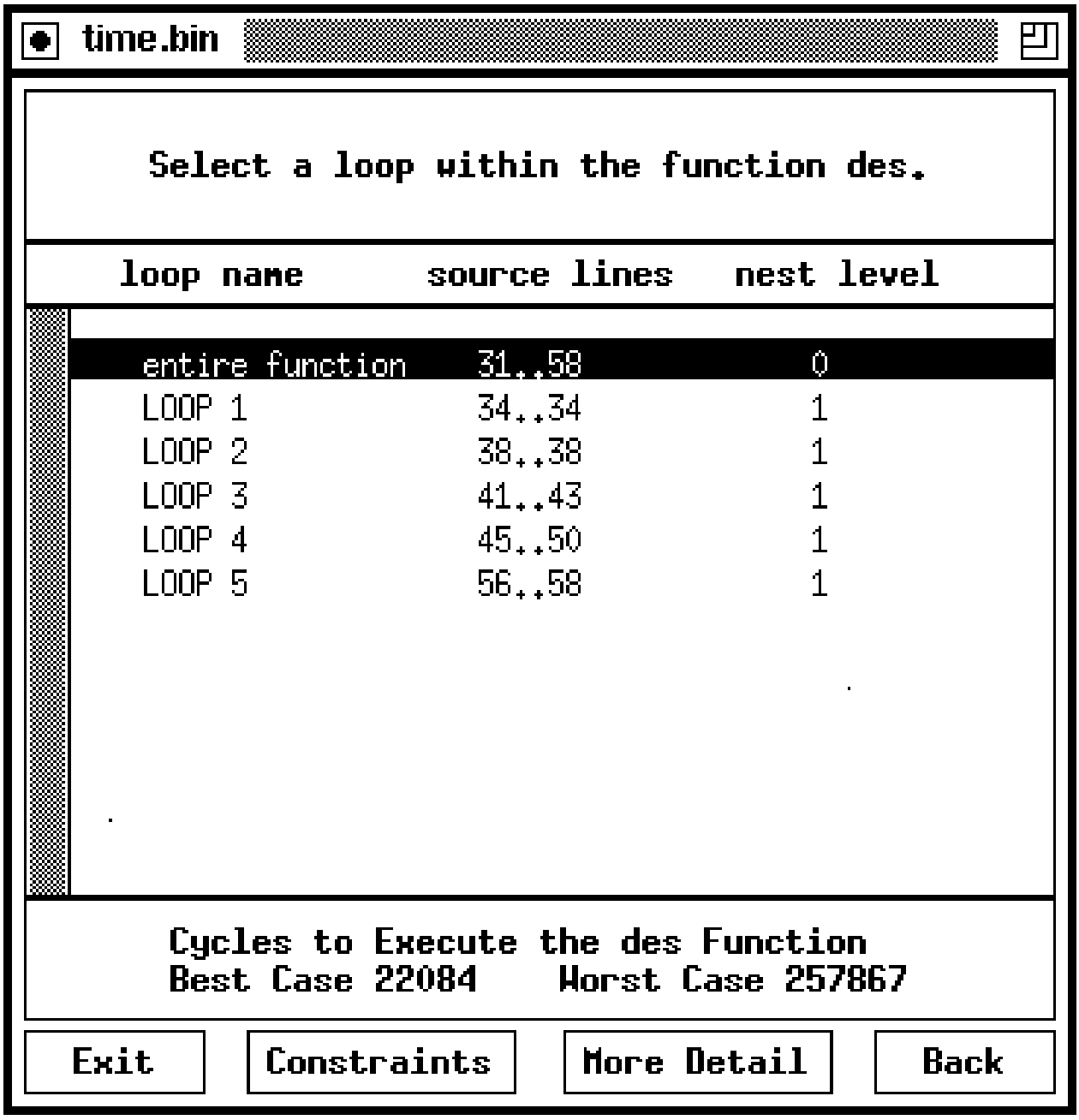
Select Path | Accept | Cancel | Clear All

Assembly Code Window

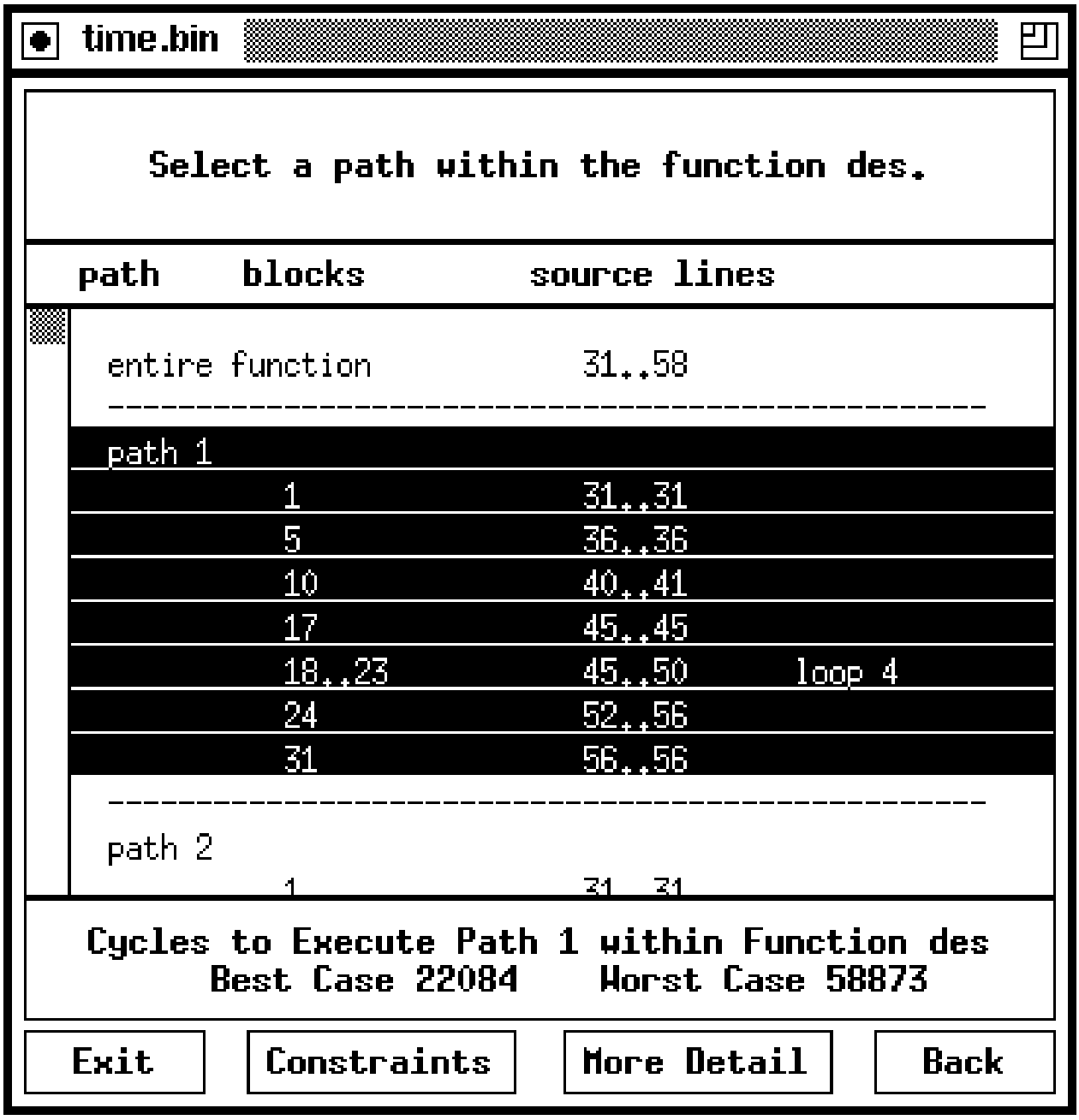
Methods for Selecting Code Portions

- Three ways to select code portions for timing predictions.
  - Constraints Window Selection: The user can quickly access the portions of the program specified in the source code timing constraints.
  - Main Window Selection: The user can make very fine-grain level requests.
  - Source Code Window Selection: The user can make requests very quickly.

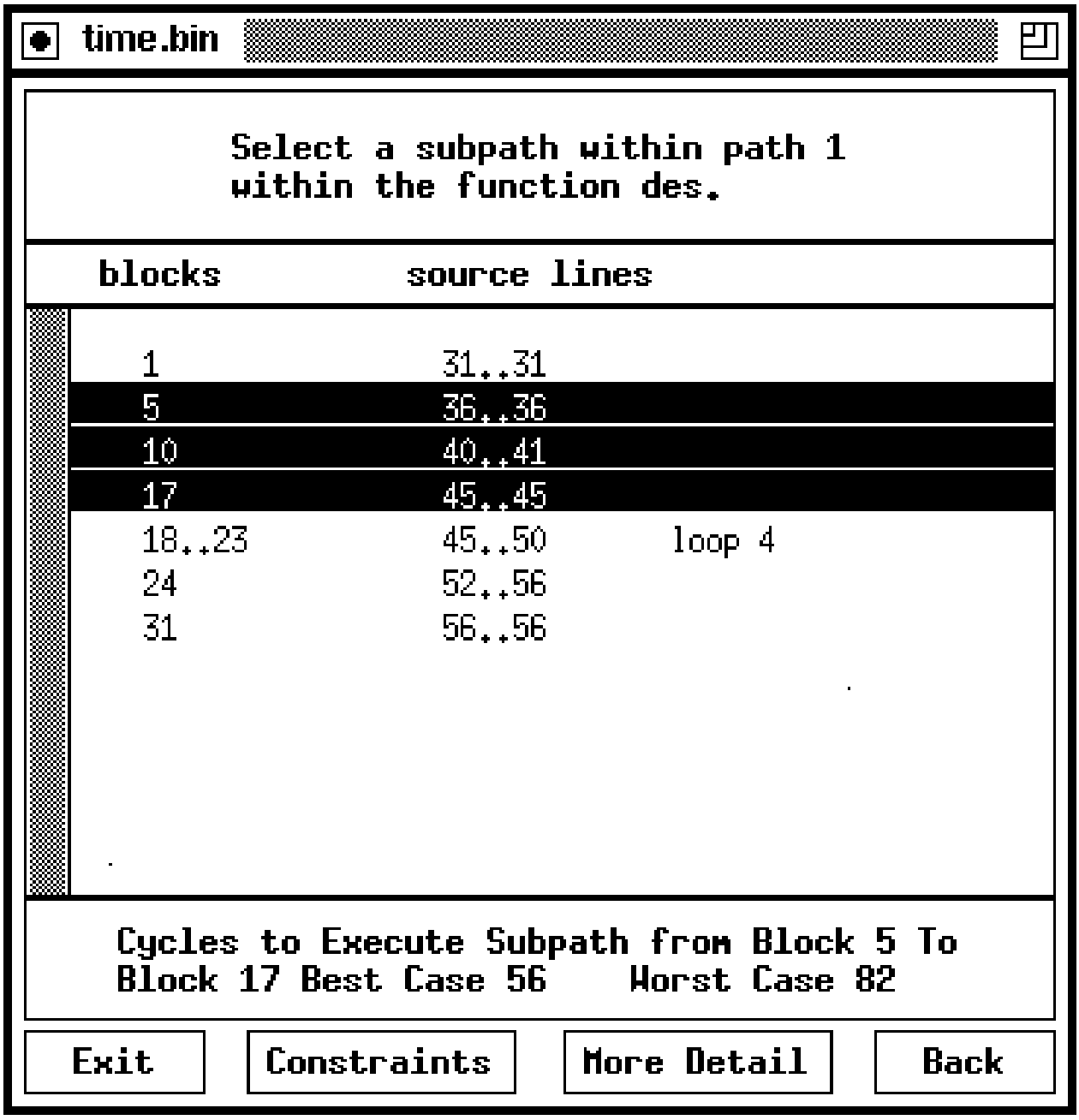
Main Window at Loop Level



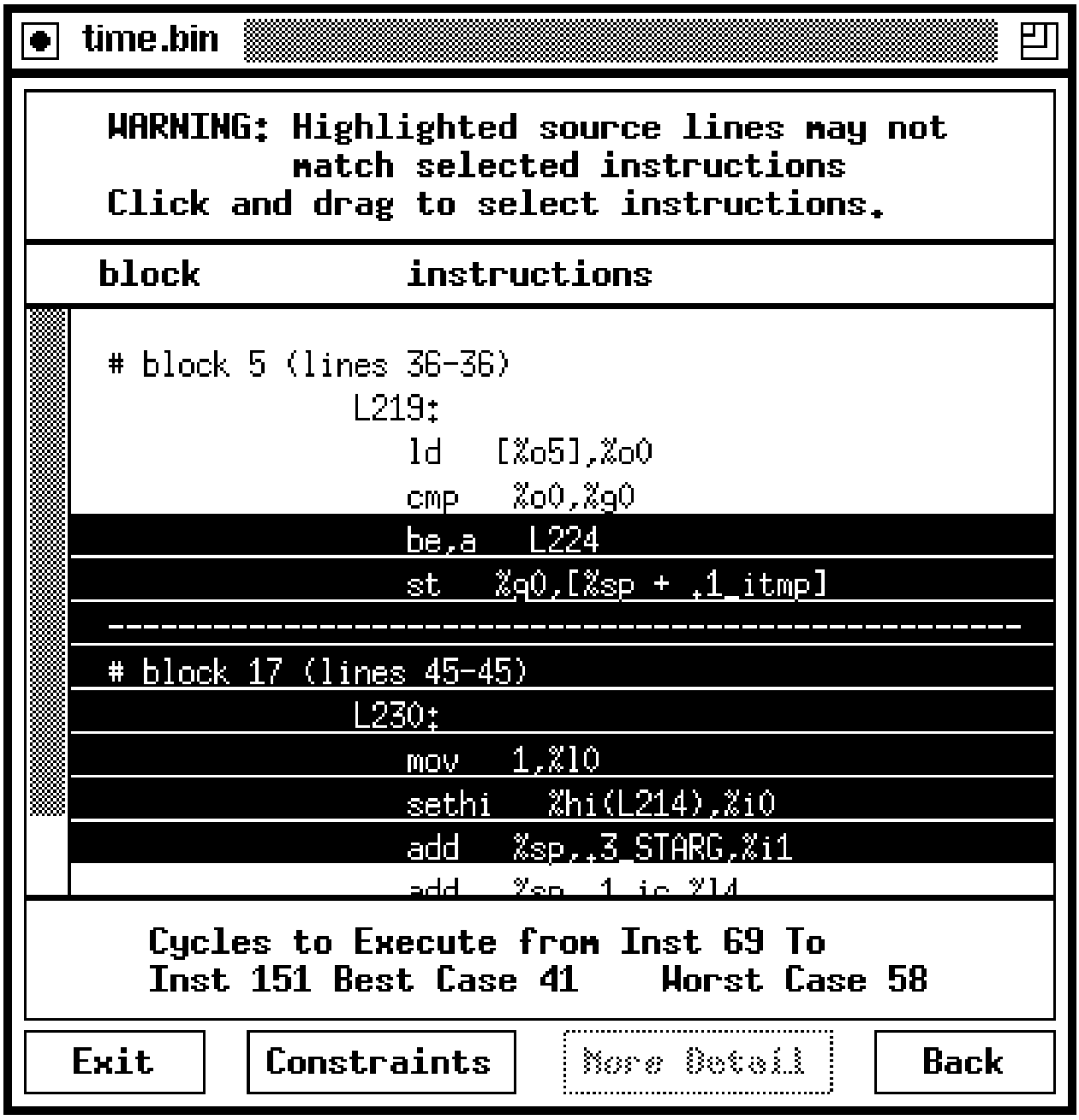
Main Window at Path Level



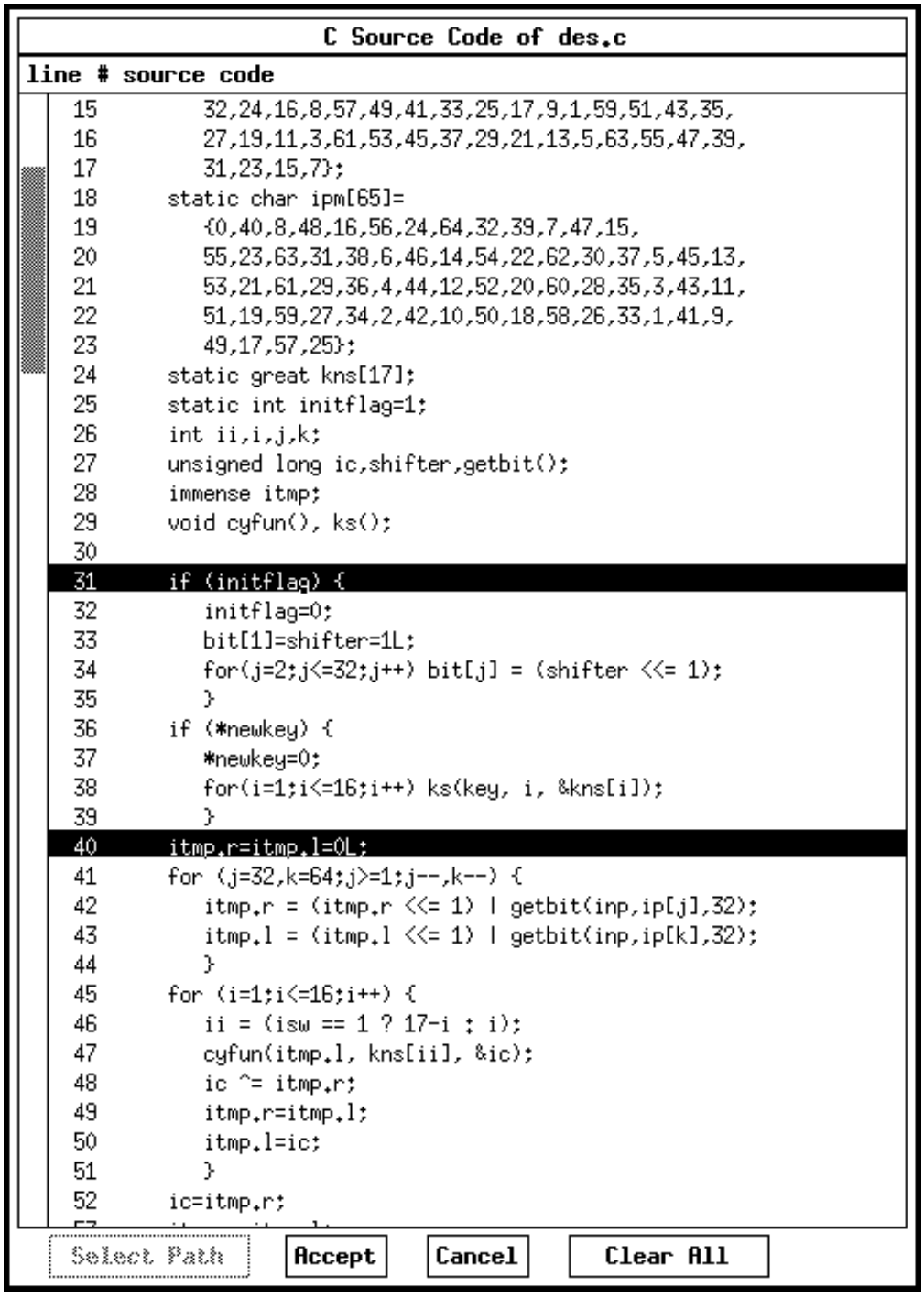
Main Window at Subpath Level



Main Window at Assembly Level



Selecting a Path via the Source Code Window



Best Case Path

C Source Code of des.c

```
line # source code
 18      static char ipm[65]=
 19         {0,40,8,48,16,56,24,64,32,39,7,47,15,
 20         55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,
 21         53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,
 22         51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,
 23         49,17,57,25};
 24      static great kns[17];
 25      static int initflag=1;
 26      int ii,i,j,k;
 27      unsigned long ic,shifter,getbit();
 28      immense itmp;
 29      void cyfun(), ks();
 30
 31      if (initflag) {
 32         initflag=0;
 33         bit[1]=shifter=1L;
 34         for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
 35         }
 36      if (*newkey) {
 37         *newkey=0;
 38         for(i=1;i<=16;i++) ks(key, i, &kns[i]);
 39         }
 40      itmp.r=itmp.l=0L;
 41      for (j=32,k=64;j>=1;j--,k--) {
 42         itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
 43         itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
 44         }
 45      for (i=1;i<=16;i++) {
 46         ii = (isw == 1 ? 17-i : i);
 47         cyfun(itmp.l, kns[ii], &ic);
 48         ic ^= itmp.r;
 49         itmp.r=itmp.l;
 50         itmp.l=ic;
 51         }
 52      ic=itmp.r;
 53      itmp.r=itmp.l;
 54      itmp.l=ic;
 55      (*out).r=(*out).l=0L;
```

Select Path | Accept | Cancel | Clear All

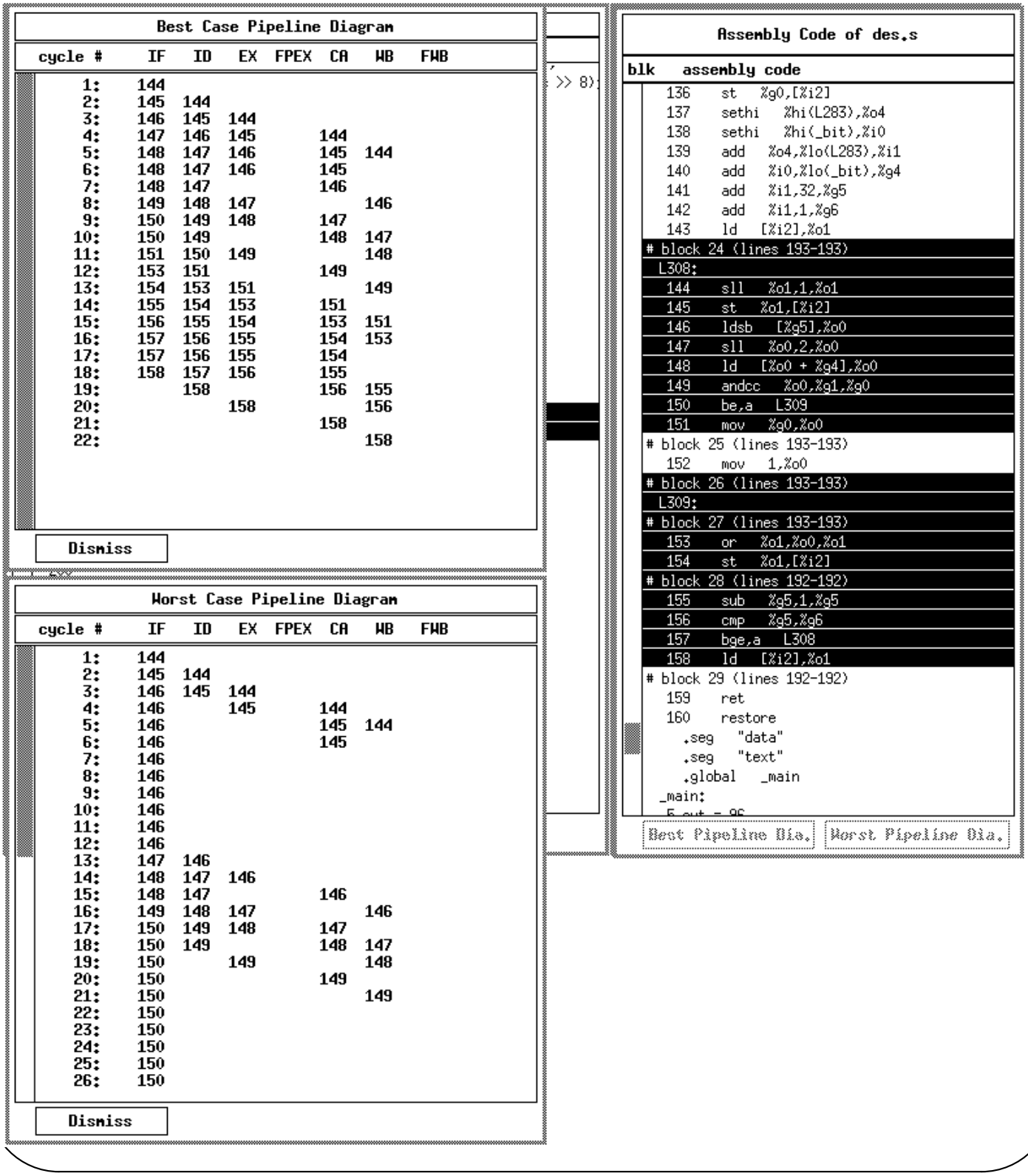
Worst Case Path

C Source Code of des.c

```
line # source code
  18      static char ipm[65]=
  19         {0,40,8,48,16,56,24,64,32,39,7,47,15,
  20         55,23,63,31,38,6,46,14,54,22,62,30,37,5,45,13,
  21         53,21,61,29,36,4,44,12,52,20,60,28,35,3,43,11,
  22         51,19,59,27,34,2,42,10,50,18,58,26,33,1,41,9,
  23         49,17,57,25};
  24      static great kns[17];
  25      static int initflag=1;
  26      int ii,i,j,k;
  27      unsigned long ic,shifter,getbit();
  28      immense itmp;
  29      void cyfun(), ks();
  30
  31      if (initflag) {
  32         initflag=0;
  33         bit[1]=shifter=1L;
  34         for(j=2;j<=32;j++) bit[j] = (shifter <<= 1);
  35         }
  36      if (*newkey) {
  37         *newkey=0;
  38         for(i=1;i<=16;i++) ks(key, i, &kns[i]);
  39         }
  40      itmp.r=itmp.l=0L;
  41      for (j=32,k=64;j>=1;j--,k--) {
  42         itmp.r = (itmp.r <<= 1) | getbit(inp,ip[j],32);
  43         itmp.l = (itmp.l <<= 1) | getbit(inp,ip[k],32);
  44         }
  45      for (i=1;i<=16;i++) {
  46         ii = (isw == 1 ? 17-i : i);
  47         cyfun(itmp.l, kns[ii], &ic);
  48         ic ^= itmp.r;
  49         itmp.r=itmp.l;
  50         itmp.l=ic;
  51         }
  52      ic=itmp.r;
  53      itmp.r=itmp.l;
  54      itmp.l=ic;
  55      (*out).r=(*out).l=0L;
```

Select Path | Accept | Cancel | Clear All

Displaying Pipeline Diagrams



Implementation

- The X Toolkit and Xlib libraries were used.
- Timing Tree
  - — Best and worst case predictions for multiple instances.
  - — Predictions for functions and loops versus paths and subpaths.

Future Work

- Supporting highlighting and selections of portions of source lines.
- Splitting a loop into sections when there are too many paths.
- Allowing assertions in the source code.
  - — loop iterations
  - — data dependencies

Conclusions

- Friendly interface for assisting programmers in the analysis of timing constraints.
  - Three methods for selecting program portions for predictions are supported.
  - Correspondence between source and machine code levels is shown.
  - Users can only select portions for which timing bounds can be obtained.
- Advantages of both high level and low level timing analysis are achieved.