Supporting User-Friendly Analysis of Timing Constraints

by

Lo Ko and David Whalley
Florida State University

and

Marion Harmon
Florida A&M University

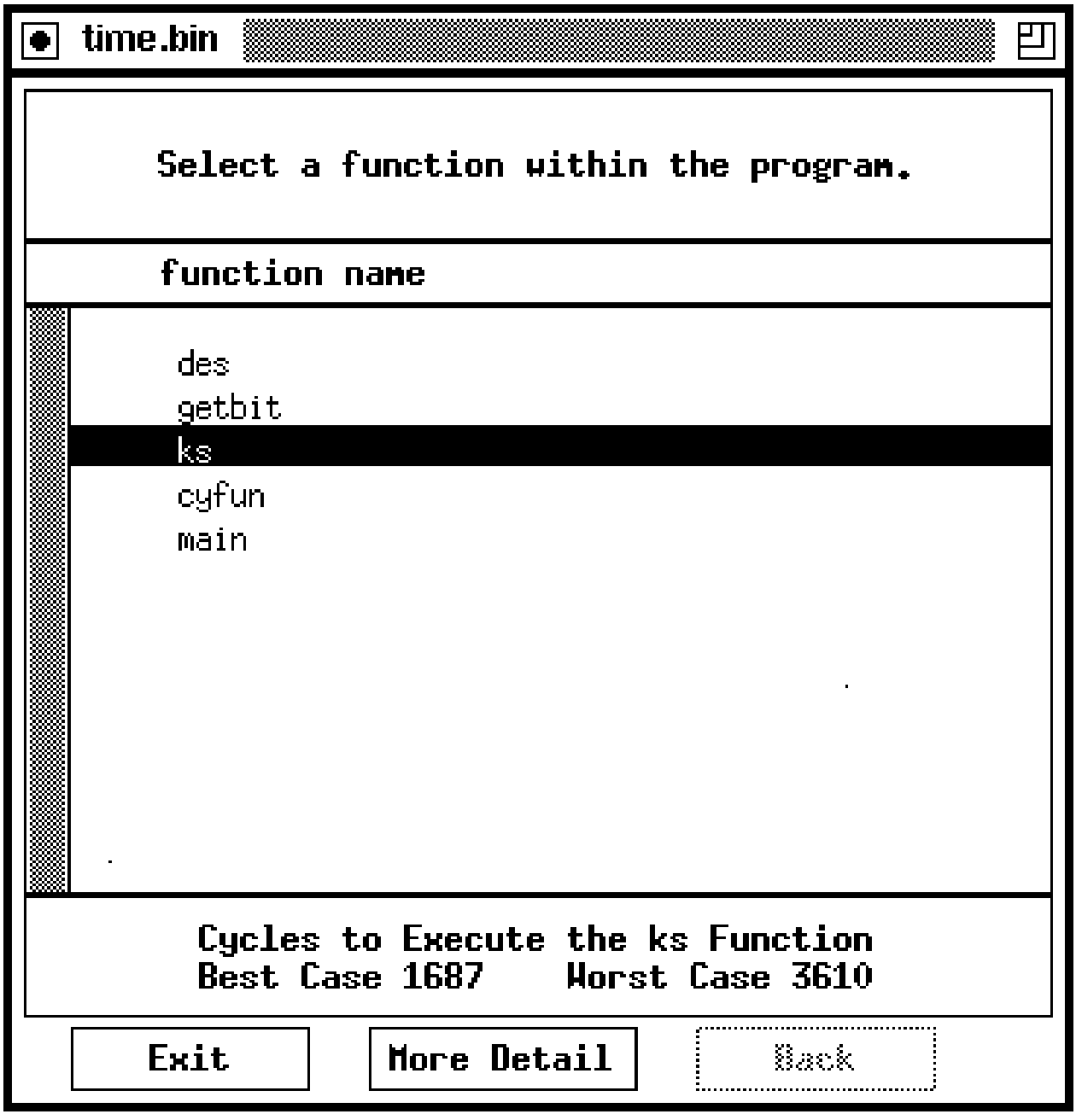
The Problem

- 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 request and understand timing predictions associated with the more accurate low-level analysis?

Goals

- A user should be able to quickly select a portion of the program for timing prediction.
- The user should only be allowed to select portions of the program for which timing bounds can be obtained.
- The corresponding portions of the source code and machine code levels of the program selected by the user for timing prediction should be depicted.

Main Window at Function Level



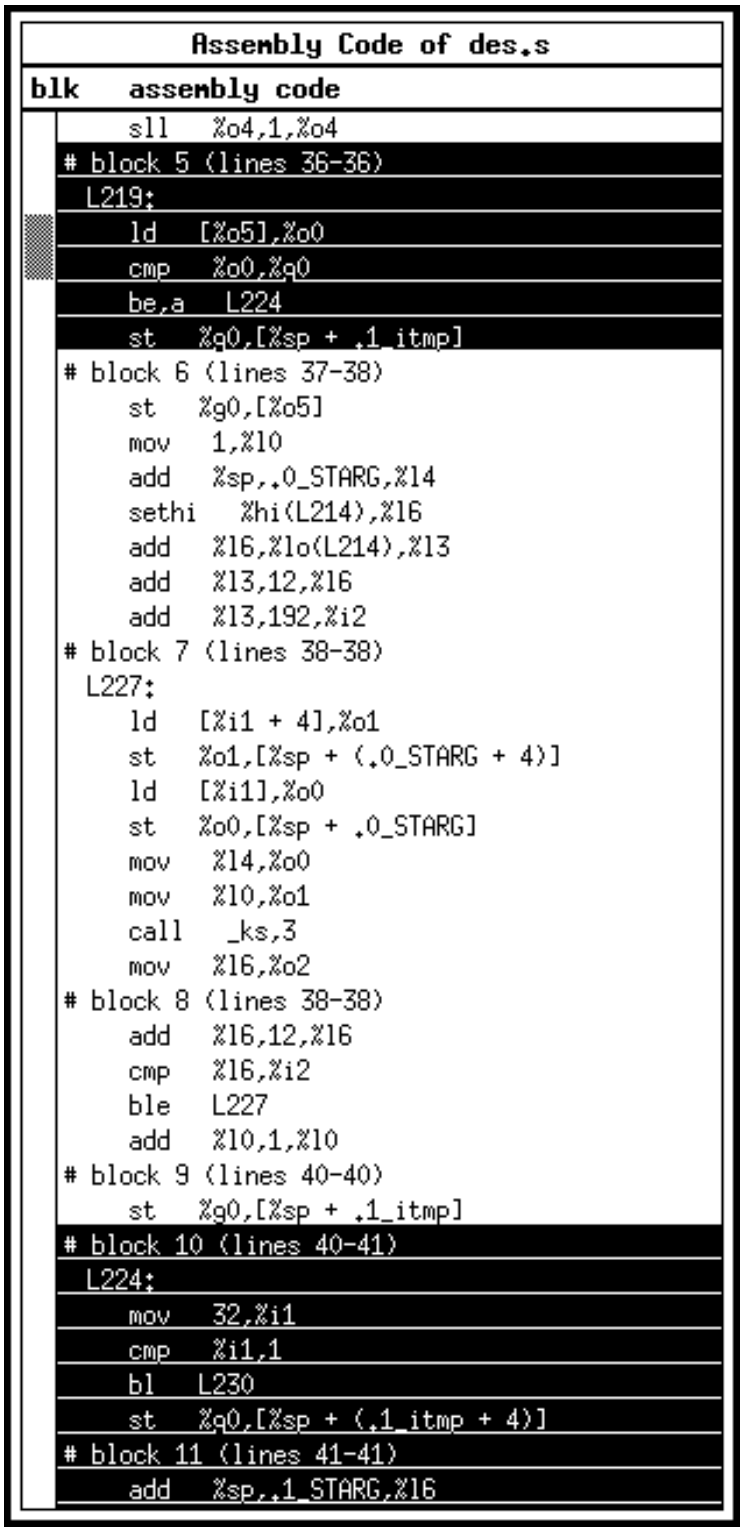
Source Code Window

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

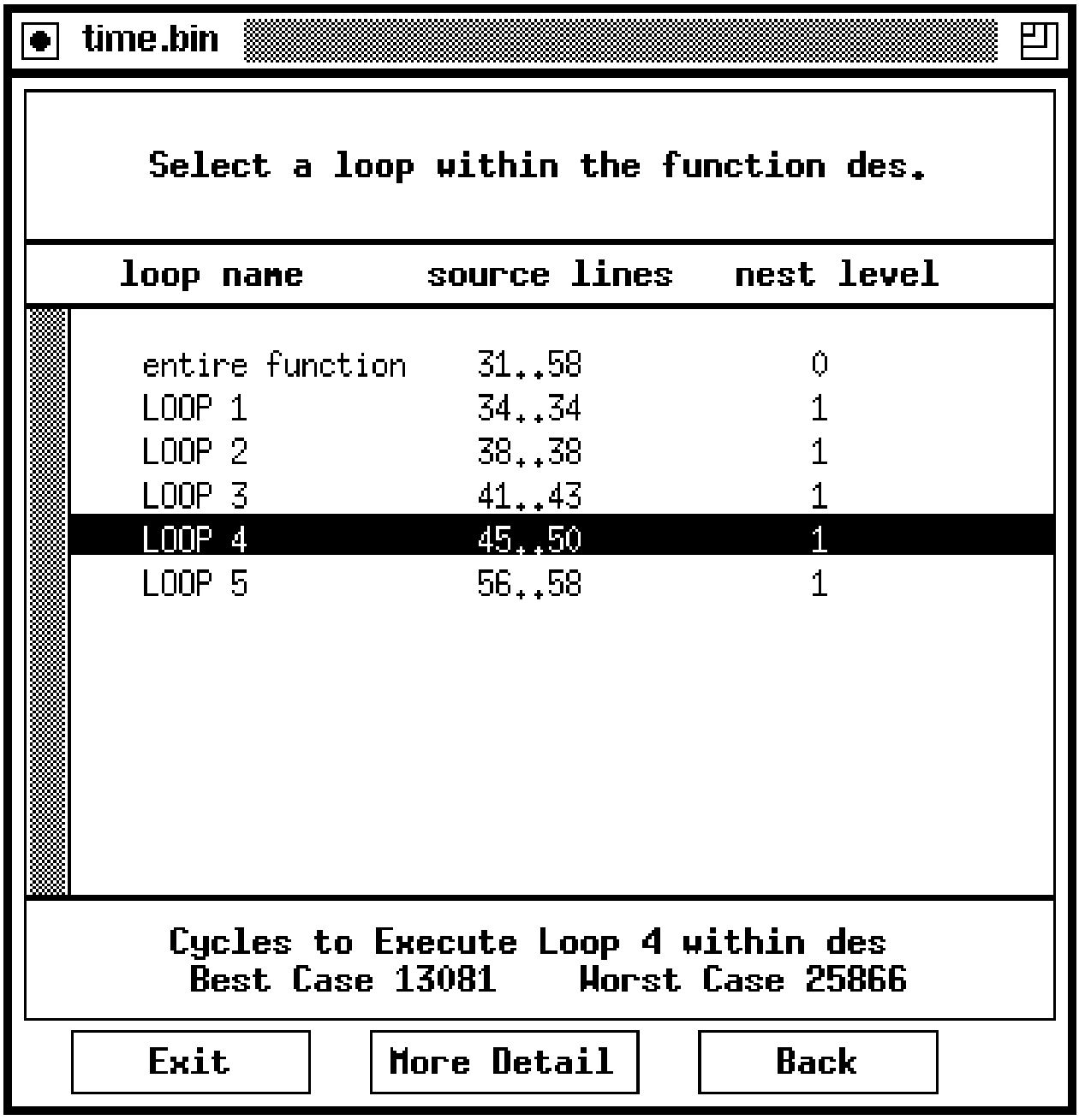
Assembly Code Window



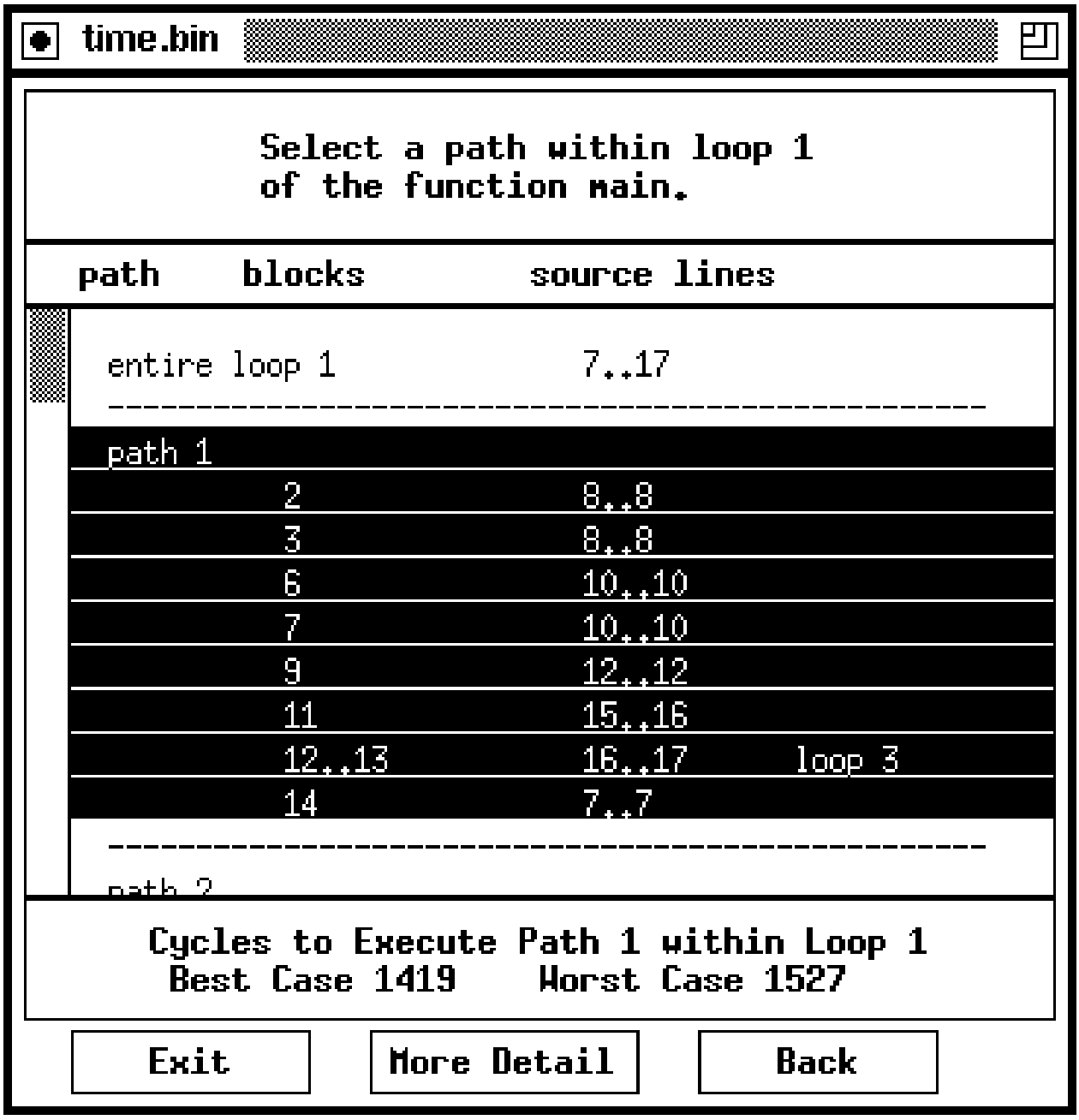
Methods for Selecting Code Portions

- Two ways to select code portions for timing predictions.
  - Main Window Selection: The user can make very fine grain level requests.
  - Source Code Window Selection: The user can make requests very quickly.

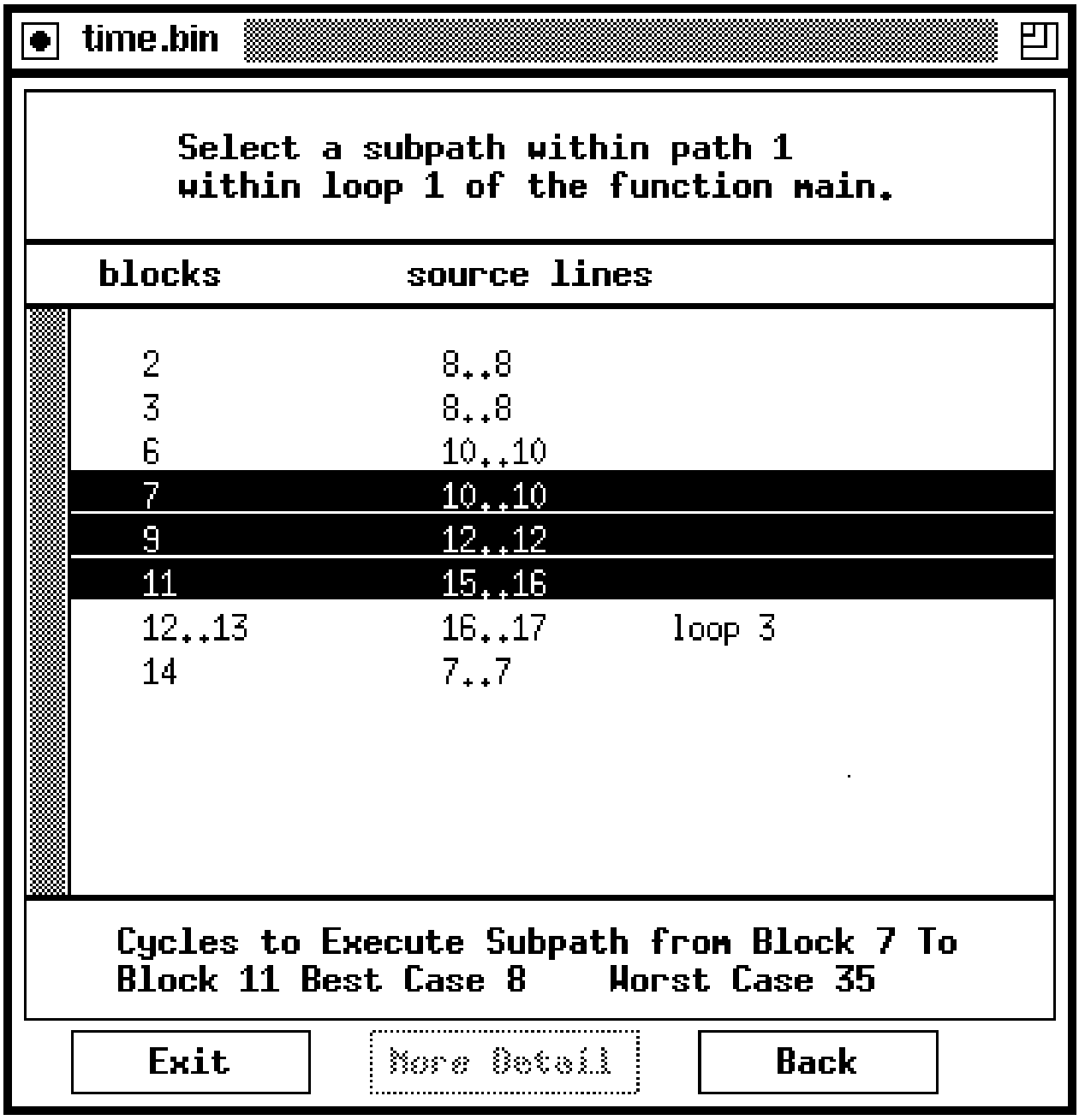
Main Window at Loop Level



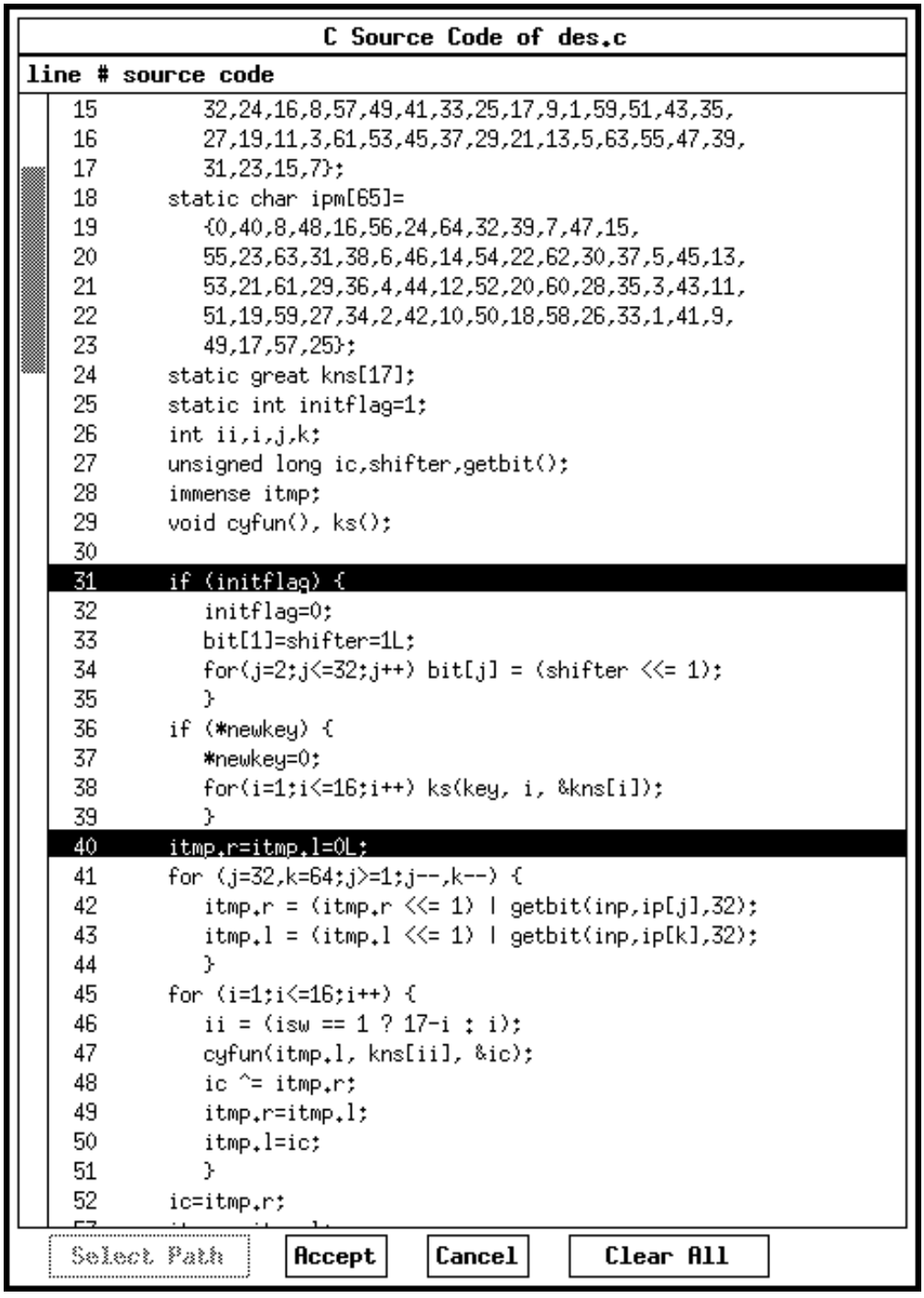
Main Window at Path Level



Main Window at Subpath Level



Selecting a Path via the Source Code Window



Best Case Path

C Source Code of des.c

```
line # source code
15          32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,
16          27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,
17          31,23,15,7};
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;
```

Select Path | Accept | Cancel | Clear All

Worst Case Path

C Source Code of des.c

```
line # source code
  15        32,24,16,8,57,49,41,33,25,17,9,1,59,51,43,35,
  16        27,19,11,3,61,53,45,37,29,21,13,5,63,55,47,39,
  17        31,23,15,7};
  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;
```

Select Path | Accept | Cancel | Clear All

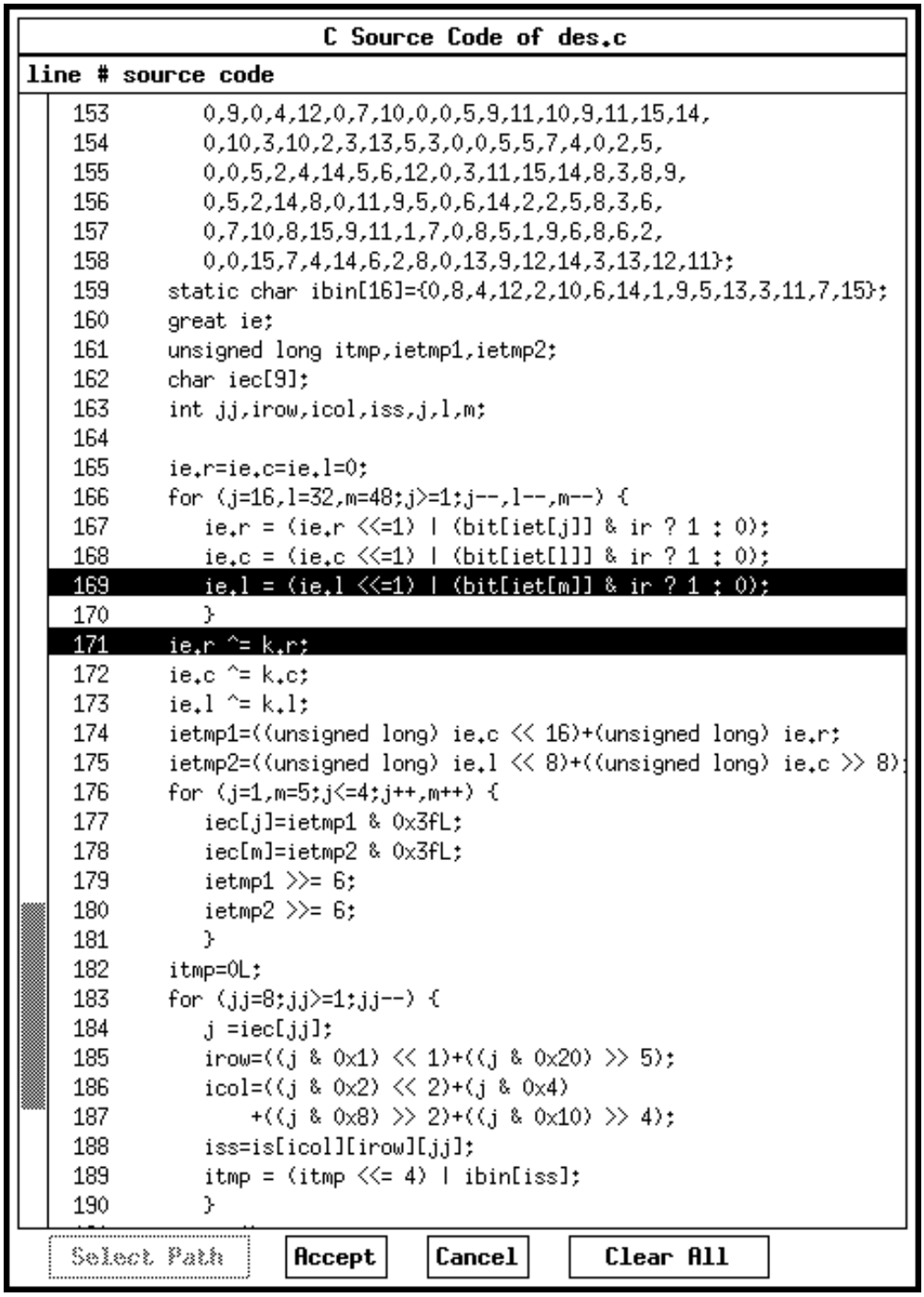
Selecting an Infeasible Path

C Source Code of des.c

```
line # source code
  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  }
  61
  62  unsigned long getbit(source,bitno,nbits)
  63  immense source;
  64  int bitno,nbits;
  65  {
  66     if (bitno <= nbits)
  67        return bit[bitno] & source.r ? 1L : 0L;
  68     else
  69        return bit[bitno-nbits] & source.l ? 1L : 0L;
  70  }
  71
  72  void ks(key,n,kn)
  73  immense key;
  74  great *kn;
  75  int n;
  76  {
  77     static immense icd;
  78     static char ipc1[57]={0,57,49,41,33,25,17,9,1,58,50,
  79        42,34,26,18,10,2,59,51,43,35,27,19,11,3,60,
  80        52,44,36,63,55,47,39,31,23,15,7,62,54,46,38,
  81        30,22,14,6,61,53,45,37,29,21,13,5,28,20,12,4};
  82     static char ipc2[49]={0,14,17,11,24,1,5,3,28,15,6,21,
  83        10,23,19,12,4,26,8,16,7,27,20,13,2,41,52,31,
  84        37,47,55,30,40,51,45,33,48,44,49,39,56,34,
  85        53,46,42,50,36,29,32};
```

Select Path | Accept | Cancel | Clear All

Selecting a Single Path



Expanded Selected Path

C Source Code of des.c

line # source code

```
153        0,9,0,4,12,0,7,10,0,0,5,9,11,10,9,11,15,14,
154        0,10,3,10,2,3,13,5,3,0,0,5,5,7,4,0,2,5,
155        0,0,5,2,4,14,5,6,12,0,3,11,15,14,8,3,8,9,
156        0,5,2,14,8,0,11,9,5,0,6,14,2,2,5,8,3,6,
157        0,7,10,8,15,9,11,1,7,0,8,5,1,9,6,8,6,2,
158        0,0,15,7,4,14,6,2,8,0,13,9,12,14,3,13,12,11};
159     static char ibin[16]={0,8,4,12,2,10,6,14,1,9,5,13,3,11,7,15};
160     great ie;
161     unsigned long itmp,ietmp1,ietmp2;
162     char iec[9];
163     int jj,irow,icol,iss,j,l,m;
164
165     ie.r=ie.c=ie.l=0;
166     for (j=16,l=32,m=48;j>=1;j--,l--,m--) {
167        ie.r = (ie.r <<=1) | (bit[iet[j]] & ir ? 1 : 0);
168        ie.c = (ie.c <<=1) | (bit[iet[l]] & ir ? 1 : 0);
169        ie.l = (ie.l <<=1) | (bit[iet[m]] & ir ? 1 : 0);
170        }
171     ie.r ^= k.r;
172     ie.c ^= k.c;
173     ie.l ^= k.l;
174     ietmp1=((unsigned long) ie.c << 16)+(unsigned long) ie.r;
175     ietmp2=((unsigned long) ie.l << 8)+((unsigned long) ie.c >> 8);
176     for (j=1,m=5;j<=4;j++,m++) {
177        iec[j]=ietmp1 & 0x3fL;
178        iec[m]=ietmp2 & 0x3fL;
179        ietmp1 >>= 6;
180        ietmp2 >>= 6;
181        }
182     itmp=0L;
183     for (jj=8;jj>=1;jj--) {
184        j =iec[jj];
185        irow=((j & 0x1) << 1)+((j & 0x20) >> 5);
186        icol=((j & 0x2) << 2)+(j & 0x4)
187            +((j & 0x8) >> 2)+((j & 0x10) >> 4);
188        iss=is[icol][irow][jj];
189        itmp = (itmp <<= 4) | ibin[iss];
190        }
```

Select Path | Accept | Cancel | Clear All

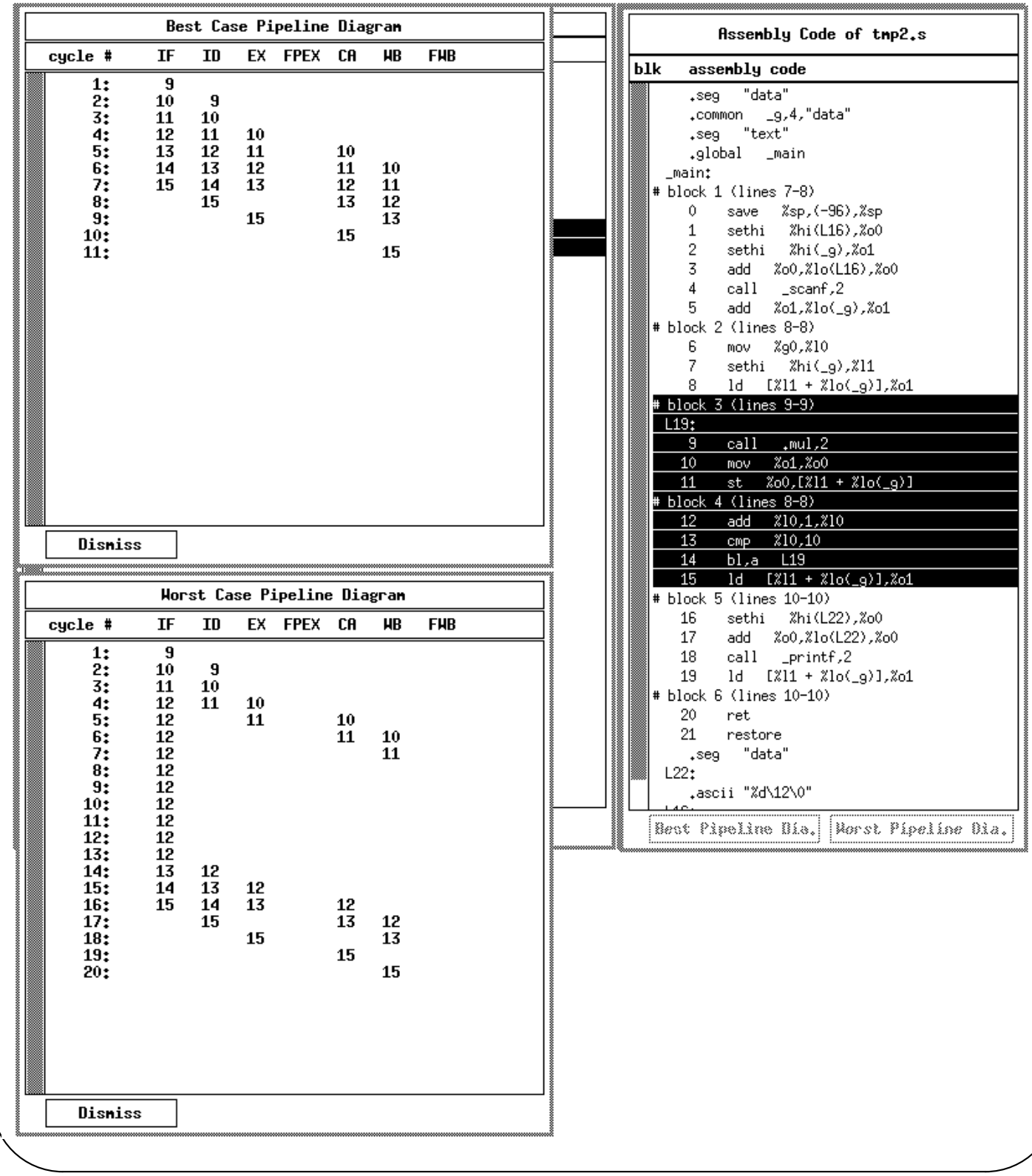
Implementation

- 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 partial highlighting of source lines.
- Displaying pipeline diagrams for subpaths.
- Permitting finer grain level of requests for timing predictions.
- Allowing assertions in the source code.
  - — loop iterations
  - — timing constraints

Displaying Pipeline Diagrams



Conclusions

- Friendly interface for assisting programmers in the analysis of timing constraints.
  - — Two 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.