# Xdlx Users Guide

## Aim

The aim of this document is to describe the facilities available in the DLX machine simulator, xdlx. If you have never used xdlx before, you should work through the document *Introduction to dlxasm and xdlx* first. It introduces you to xdlx in a step by step fashion. This document, on the other hand, is more of a reference for you to to find out about particular facilities you might want to use.

## Starting xdlx

You can start xdlx on your workstation by entering the command:

`xdlx` *file*

This starts up the simulator, and loads the memory with the contents of the memory image file *file*. The file argument is optional; if you omit it, the memory is initialized to all zeros.

The simulator creates a new window on your display, divided into four main panels, as shown in Figure 1.

- ***Registers***: The top right quarter of the window contains the 32 general purpose registers (r0 to r31) and the program counter (PC). To the right of each register is a box that shows the display format of the register.
- ***Code panel***: The top left quarter of the window shows the contents of memory represented as DLX instructions. Note that all memory locations are shown here, irrespective of whether they are actually part of a program or not. The location pointed to

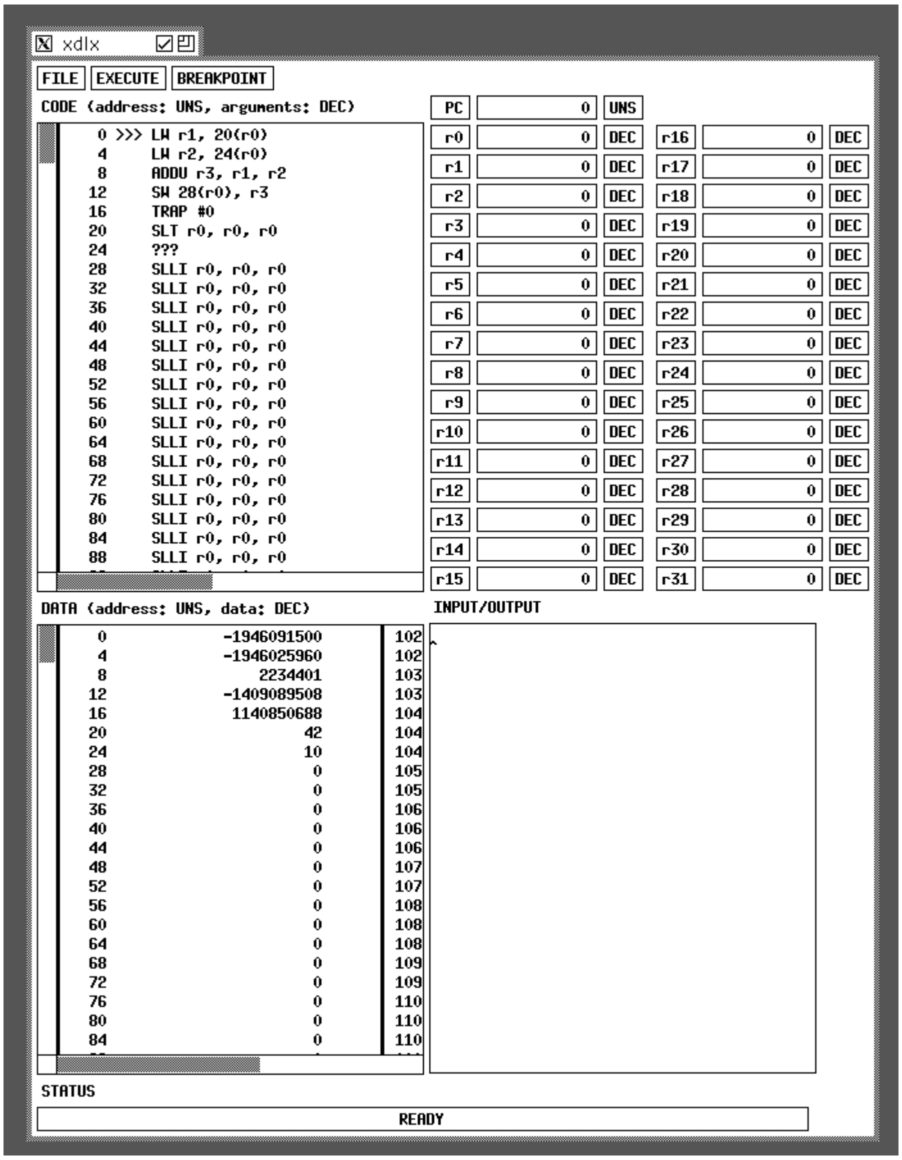


**Figure 1.** The xdlx window

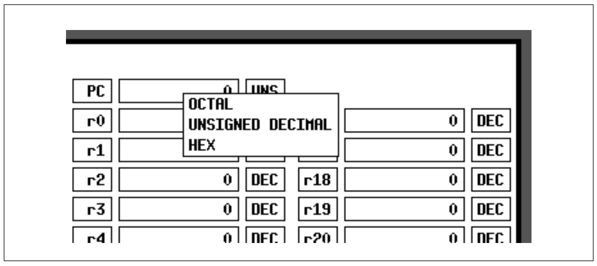


**Figure 2.** Changing the PC display format.

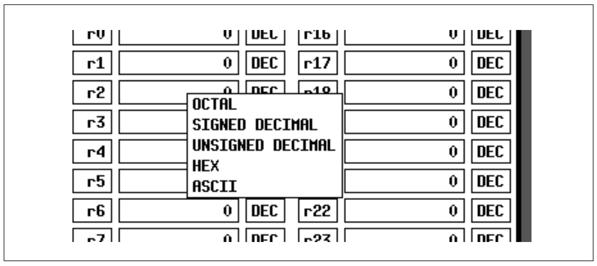


**Figure 3.** Changing the register display format.

by the PC is indicated with the PC marker (“>>>”). Above the panel, a header line shows you the display formats used for addresses and instruction arguments.

- ***Data panel***: The bottom left quarter of the window shows the contents of memory interpreted as data. Above the panel, a header line shows you the display formats used for addresses and data.
- ***Input/output panel***: The bottom right quarter of the window is a panel for entering input and displaying output from a program.

Along the top of the xdlx window are a number of menu boxes, headed *FILE*, *EXECUTE* and *BREAKPOINT*, for controlling the way a program is run.

At the bottom of the window is a *STATUS* line, in which xdlx displays informative messages as menu commands are selected and a program is executed.

## Changing display formats

Information in xdlx registers and panels can be displayed in a number of different ways, including octal (base 8), unsigned decimal, signed decimal, hexadecimal (base 16) and ASCII. The current display format for each register or panel is shown using one of the following codes:

- octal: *OCT*
- signed decimal: *DEC*
- unsigned decimal: *UNS*
- hexadecimal: *HEX*
- ASCII characters: *ASC*

### Changing the PC display format

The box to the right of the each of the registers shows the current display format for the register. If you press mouse button 3 over the box, a menu appears. For the PC, you can choose one of octal, unsigned decimal or hexadecimal (see Figure 2). For the general purpose registers, you can choose any of the five display formats mentioned above (see Figure 3). When you select one of these formats, the label in the box to the right of the register is changed accordingly, and the register is displayed in the selected format. Note that this does not alter the content of the register, only the way the bits are displayed

### Changing code panel display format

You can select the display format used for addresses in the code panel by pressing button 3 over the header above the code panel. A menu appears, allowing you to

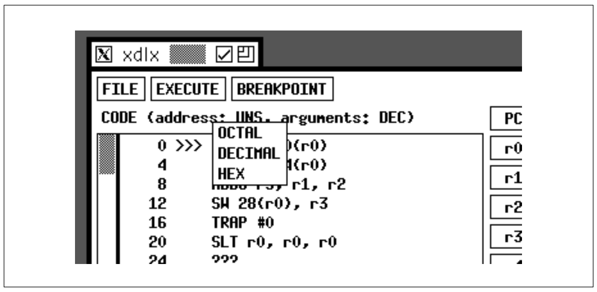


**Figure 4.** Changing the code address display format.

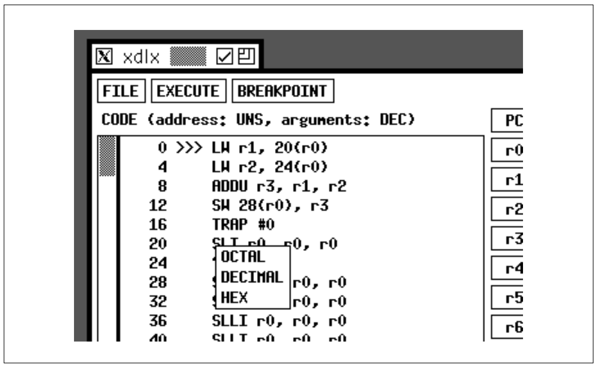


**Figure 5.** Changing the code argument display format.

choose octal, unsigned decimal or hexadecimal format (see Figure 4).

You can also change the display format used for instruction arguments by pressing mouse button 3 within the code panel. A menu appears, allowing you to choose octal, decimal or hexadecimal (see Figure 5). When decimal display format is selected, arguments are shown in signed or unsigned format, depending on the opcode. Displacements and arguments for signed integer instructions are shown in signed format, whereas arguments for unsigned integer instructions are shown in unsigned format.

### Changing data panel display format

You can change the display format for addresses in the data panel by pressing button 3 over the data panel header. This is similar to changing the address format for the code panel, described above.

There are two aspects to the display format for data within the data panel. Firstly, the data can be displayed in any of the five formats previously mentioned. Secondly, the data can be shown as separate bytes, as halfwords or as whole words. Pressing mouse button 3 over the data panel brings up a combined menu (see Figure 6). Use the menu once to select the format, then again to select the data size.

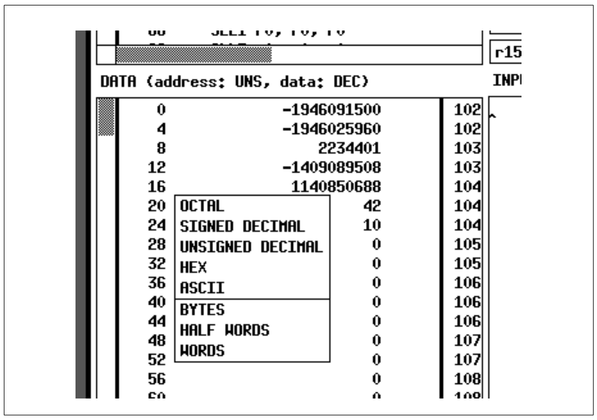


**Figure 6.** Changing the data panel display format.

## Scrolling the panels

The code and data panels each have scroll bars. The grey block in the bar represents the proportion of the memory that is visible in the panel. The position of the block represents the position in the memory that is shown in the panel. The input/output panel also gets scroll bars when there is more text in the panel than can be displayed in the area available.

You scroll a panel vertically by clicking in the scroll bar on the left of the panel. Clicking with button 1 scrolls down, while clicking with button 3 scrolls up. If you click with button 2, the grey box is positioned where you clicked in the bar, and the panel is scrolled to the corresponding position in the memory. You can also press and drag button 2 to scroll the panel as you move the mouse.

Memory locations are organized in the panels in columns. The first column starts at address 0, and goes down to address 1023. Subsequent blocks of 1024 addresses are shown in columns further to the right in each of the code and data panels, separated by vertical black lines. (The column divider in the data panel is visible in Figure 1.) If you need to see these columns, you can scroll the panel horizontally using the scroll bar at the bottom of the panel. Clicking with button 1 scrolls to the right, while clicking with button 3 scrolls to the left. Clicking or dragging with button 2 scrolls the panel horizontally to an absolute position.

## Editing values

You can edit the value in a register or the PC by moving the mouse pointer over the register value, and double-clicking with mouse button 1. A small dialogue box pops up containing the value of the register. You can edit the value using the same character editing keystrokes as you use in xedit. The value you end up with must be valid for the display format currently being used for the register. When you have completed your changes, either click with mouse button 1 on the *Ok* box or press the *Return* key to update the register with the new value. Alternatively, you can click on the *Cancel* box to discard any edits you have made, and leave the register unchanged. Note that register r0 cannot be edited in this way, since it is always 0.

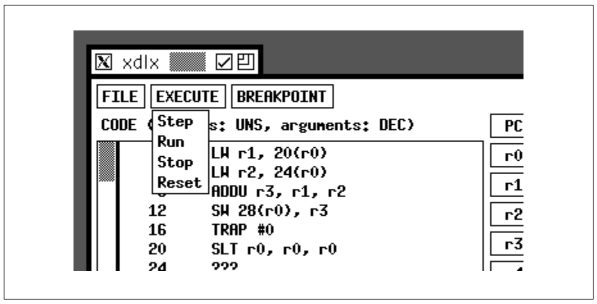


**Figure 7.** The *EXECUTE* menu.

You can also edit the values in memory locations in the same way, by moving the mouse pointer over the location in the data panel and double-clicking with mouse button 1.

Note that you cannot edit instructions in memory using the code panel. If you need to change the program, you should edit the assembler language source file, reassemble the program, then load the new memory image file using the *Open* command (see below).

## Executing the program

The PC register contains the address of the next instruction to be executed by xdlx. This instruction is indicated with the PC marker (">>>") in the code panel. Whenever the PC changes, this marker moves correspondingly. You can control execution of the program using the commands in the *EXECUTE* menu, by clicking on the box with mouse button 3 (see Figure 7).

The *Step* command causes xdlx to fetch and execute the one instruction pointed to by the PC, then stop. The register and memory displays are updated to reflect any values that may have been changed by the instruction, and the PC marker is moved to the next instruction to be executed.

The *Run* command causes xdlx to run the program without stopping between instructions. Running starts at the instruction pointed to by the PC, and continues until

- the program terminates (by executing a trap 0 instruction),
- a breakpoint is reached (see next section),
- an error occurs, or
- you stop it by selecting the *Stop* command.

While the program is running, the register and memory contents are not updated on the screen. However, the screen is updated when the program stops for any of the above reasons.

The *Reset* command causes xdlx to reset the contents of all registers to 0, and to reload the memory from the memory image file. You can use this command if you want to start the program again from scratch. Note that the *Reset* command does not clear any breakpoints that may be in effect.

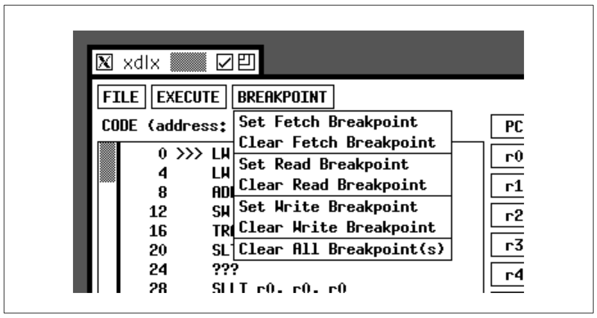


**Figure 8.** The *BREAKPOINT* menu.

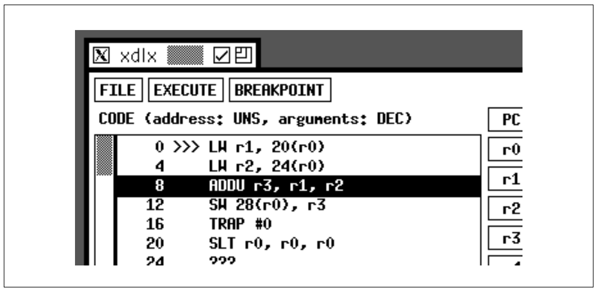


**Figure 9.** Selecting an instruction.

## Using breakpoints

A breakpoint is a location in memory that causes xdlx to stop execution when accessed. There are three kinds of breakpoints:

- fetch breakpoint: xdlx stops when the location is fetched as an instruction to be executed
- read breakpoint: xdlx stops when the location is to be read by a load instruction
- write breakpoint: xdlx stops when the location is to be written by a store instruction

In all cases, when the breakpoint is reached, xdlx stops before the operation takes place, leaves the PC pointing to the instruction, and displays a message in the status line. You can then examine contents of registers and memory, and continue execution with the *Step* or *Run* commands. The breakpoint stays in effect on the location, so if it is accessed again, xdlx will stop again.

You can set and clear breakpoints on memory locations using the commands in the *BREAKPOINT* menu, by clicking on the box with mouse button 3 (see Figure 8).

### Setting and clearing fetch breakpoints

To set a fetch breakpoint on an instruction takes two stages. Firstly, you select the instruction by clicking with mouse button 1 in the code panel. The instruction is highlighted as shown in Figure 9. Next, you select the *Set Fetch Breakpoint* command. The location is marked by drawing an outline border around it, as shown in Figure 10.

To clear a fetch breakpoint on an instruction, firstly select the instruction by clicking on it with mouse button 1 in the code panel, then choose *Clear Fetch Breakpoint* from the *BREAKPOINT* menu. The outline border is removed, indicating that xdlx has cleared the breakpoint.

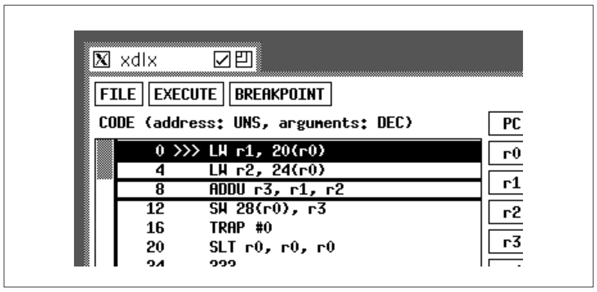


**Figure 10.** A fetch breakpoint set.

### Setting and clearing read and write breakpoints

To set a read or write breakpoint on a memory location, you firstly select the location in the data panel by clicking on it with mouse button 1. The location is highlighted. Next, select the *Set Read Breakpoint* or *Set Write Breakpoint* command from the *BREAKPOINT* menu. The location in the data panel is outlined.

When you use these commands, the breakpoint is set on the *byte* at location selected. This is so even if the data panel display format is halfword or word. In these cases, the breakpoint is only set on the first byte in the halfword or word.

When a byte with a read breakpoint set is read during program execution, xdlx stops execution. This includes reading the byte as part of a halfword or word. For example, if the byte at location 22 has a read breakpoint set, and the program reads a word from location 20, xdlx stops. The same behaviour applies during writes to memory when write breakpoints are set. On the other hand, if a halfword or word contains a byte with a breakpoint set, and some other byte in that halfword or word is accessed, xdlx does not stop execution.

When the data panel displays memory in halfword or word format, halfwords or words that contains any bytes with breakpoints set are highlighted. If you select one of these locations and choose the *Clear Read Breakpoint* or *Clear Write Breakpoint* command, the command is only applied to the first byte in the halfword or word. If other bytes also have breakpoints set, you will see a message to that effect in the status line at the bottom of the window. You can see which particular bytes have breakpoints by changing the display format to display individual bytes.

### Clearing all breakpoints

You can clear all fetch, read and write breakpoints by choosing the *Clear All Breakpoint(s)* command from the *BREAKPOINT* menu. Choosing the *New* or *Open* command from the *FILE* menu (see below) also has the effect of clearing all breakpoints.

## Using the *FILE* menu

The commands in the *FILE* menu allow you to load and save memory image files (see Figure 11).

The *New* command clears all memory locations to zero, clears all breakpoints, and clears all registers and the PC to zero.

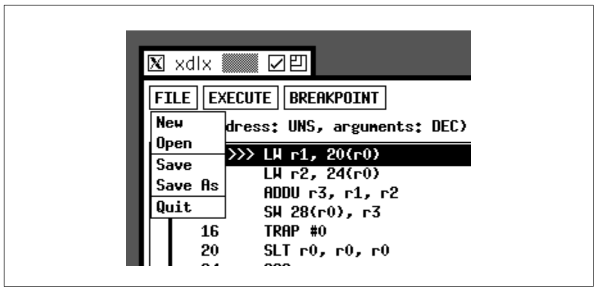


**Figure 11.** The *FILE* menu.

The *Open* command allows you to choose a memory image file to be loaded into the simulated memory. A file dialogue box pops up for you to choose the file name. When you have done so, xdlx loads the content of the file into memory, clears all breakpoints, and clears all registers and the PC to zero.

The *Save* command allows you to save changes made to the memory back into the memory image file most recently loaded. The *Save As* command allows you to save changes into a new memory image file. It pops up a dialogue box for you to choose the file name.

The *Quit* command allows you to finish using the simulator. The window is removed from the screen, and the contents of the memory are discarded.

### File dialogue box

The *Open* and *Save As* commands need you to choose a memory image file. The dialogue box they pop up for you is shown in Figure 12. You can type the name of the file in the line at the top of the box. When you click on the *Ok* box with mouse button 1, or press the *Return* key, the file name you have typed is used for the command. Alternatively, you can click on the *Cancel* button to cancel the *Open* or *Save As* command.

Below the file name line, all of the files in the current directory are listed in alphabetical order. You can scroll the list to find the name you want. If you click on one of the names, it is copied into the file name line at the top of the dialogue box. You can then edit it to form a new file name. Alternatively, if you double-click on a file name, it is used for the *Open* or *Save As* command. (This is like clicking on the name once, then clicking on the *Ok* box.) If you double-click on a directory name in the list, it is copied into the file name line, and the list is changed to display the contents of the directory. If you want to re-display the contents of the current directory, double-click on the list entry "./". If you want to see the contents of the parent directory, double-click on the list entry "../".

The dialogue box also allows you to search file names using auto-completion. If you type part of a file name, then press the *Tab* or *Space* key, xdlx will attempt to fill in the rest of the file name by looking at what files are listed. If there is more than one possibility, it only fills in as many characters as are common to them all. You can then type more characters to disambiguate the name, or press the *Space* key again to get a list of possible completions in the list box. At any time, you can click or double-click on one of the candidates listed to use that name.

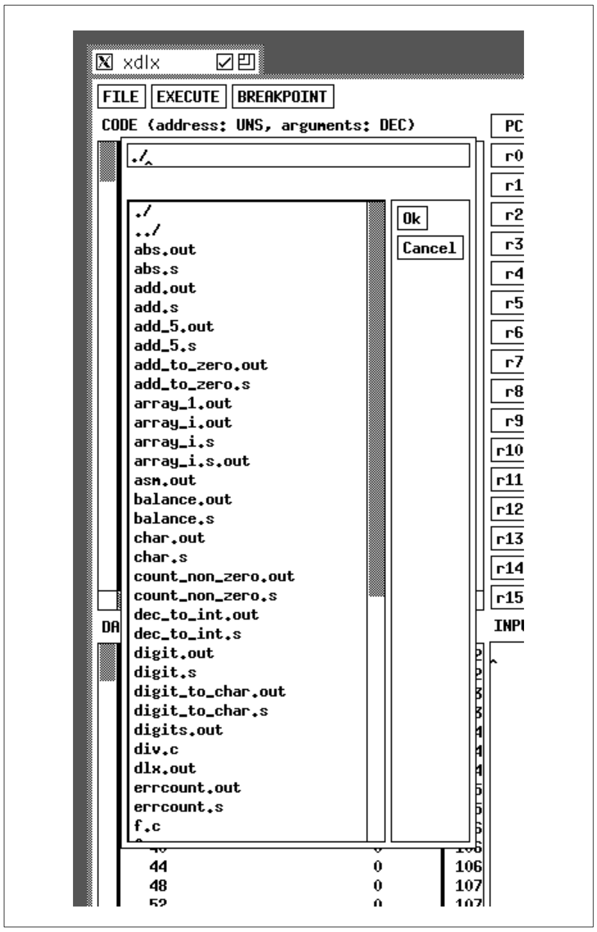


**Figure 12.** File dialog box.

## Trap instructions

Xdlx interprets *trap* instructions as system calls. The operand to the trap instruction selects which system operation to perform. Some operations require further arguments, passed in registers, or return results in registers. The systems operations implemented by xdlx are as follows:

- trap 0: halt

  Xdlx stops execution. The PC is left pointing at the trap instruction.

- trap 3: character input

  Xdlx waits for you to position the mouse pointer over the input/output panel and type an input character. The ASCII code for the character is returned in register r1.

- trap 4: character output

  Xdlx treats the content of register r1 as the ASCII code for a character, and displays it in the input/output panel. Printable characters are displayed at the current cursor location in the panel. CR causes the cursor to return to the left margin, LF causes it to move down one line, and BEL causes the workstation to beep. Other control characters are ignored.

## Error exceptions

There are a number of exceptional or error conditions that can arise during program execution. In all cases, xdlx stops execution, leaves the PC pointing to the instructions that caused the exception, and displays a message in the status line at the bottom of the window. The exceptions that are handled in this way are:

- accessing non-existent memory locations: arises when the program attempts to access a memory location at an addresses beyond the highest address of the simulated memory
- accessing mis-aligned data: arises when the program attempts to access a halfword at an odd address, or a word at an address that is not a multiple of four
- integer overflow: arises when the result of an arithmetic operation falls outside the range of representable integers
- unimplemented instruction: arises when the binary value fetched from memory does not correspond to a DLX instruction (a word displayed as “???” in the code panel)
- floating point operation: arises when a floating point instruction is fetched; xdlx does not simulate these instructions