NAME

DLXsim - Simulator and debugger for DLX assembly programs

SYNOPSIS

dlxsim

OPTIONS

[-al#] [-au#] [-dl#] [-du#] [-ml#] [-mu#]

-al# Select the latency for a floating point add (in clocks).

-au# Select the number of floating point add units.

-dl# Select the latency for a floating point divide.

-du# Select the number of floating point divide units.

-ml# Select the latency for a floating point multiply.

-mu# Select the number of floating point multiply units.

DESCRIPTION

DLXsim is an interactive program that loads DLX assembly programs and simulates the operation of a DLX computer on those programs. When **DLXsim** starts up, it looks for a file named **.dlxsim** in the user's home directory. If such a file exists, **DLXsim** reads it and processes it as a command file. **DLXsim** also checks for a **.dlxsim** file in the current directory, and executes the commands in it if the file exists. Finally, **DLXsim** loops forever reading commands from standard input and printing results on standard output.

NUMBERS

Whenever **DLXsim** reads a number, it will accept the number in either decimal notation, hexadecimal notation if the first two characters of the number are **0x** (e.g. 0x3acf), or octal notation if the first character is **0** (e.g. 0342). Two **DLXsim** commands accept only floating pointer numbers from the user; these are **fget** and **fput** and will be described later.

ADDRESS EXPRESSIONS

Many of **DLXsim**'s commands take as input an expression identifying a register or memory location. Such values are indicated with the term *address* in the command descriptions below. Where register names are acceptable, any of the names **r0** through **r31** and **f0** through **f31** may be used. The names **$0** through **$31** may also be used (instead of **r0** through **r31**), but the dollar signs are likely to cause confusion with Tcl variables, so it is safer to use **r** instead of **$**. The name **pc** may be used to refer to the program counter.

Symbolic expressions may be used to specify memory addresses. The simplest form of such an expression is a number, which is interpreted as a memory address. More generally, address expressions may consist of numbers, symbols (which must be defined in the assembly files currently loaded), the operators $*$, /, %, +, $-$, <<, >>, &, |, and $\uparrow$ (which have the same meanings and precedences as in C), and parentheses for grouping.

COMMANDS

In addition to all of the built-in Tcl commands, **DLXsim** provides the following application-specific commands:

asm *instruction* [*address*]

Treats *instruction* as an assembly instruction and returns a hexadecimal value equivalent to *instruction*. Some instructions, such as relative branches, will be assembled differently depending on where in memory the instruction will be stored. The *address* argument may be used to indicate where the instruction would be stored; if omitted, it defaults to 0.

fget *address* [*flags*]

Return the values of one or more memory locations or registers. *Address* identifies a memory location or register, and *flags*, if present, consists of a number and/or set of letters, all concatenated together. If the number is present, it indicates how many consecutive values to print (the default is 1). If flag characters are present, they have the following interpretation:

d Print values as double precision floating point numbers.

f Print values as single precision floating point numbers (default).

fput *address number* [*precision*]

Store *number* in the register or memory location given by *address*. If *precision* is **d**, the number is stored as a double precision floating point number (in two words). If *precision* is **f** or no *precision* is given, the number is stored as a single precision floating point number.

get *address* [*flags*]

Similar to **fget** above, this command is for all types except floating point. If flag characters are present, they have the following interpretation:

B Print values in binary.

b When printing memory locations, treat each byte as a separate value.

c Print values as ASCII characters.

d Print values in decimal.

h When printing memory locations, treat each halfword as a separate value.

i Print values as instructions in the DLX assembly language.

s Print values as null-terminated ASCII strings.

v Instead of printing the value of the memory location referred to by *address*, print the address itself as the value.

w When printing memory locations, treat each word as a separate value.

x Print values in hexadecimal (default).

To interpret numbers as single or double precision floating point, use the **fget** command.

go [*address*]
Start simulating the DLX machine. If *address* is given, execution starts at that memory address. Otherwise, it continues from wherever it left off previously. This command does not complete until simulated execution stops. The return value is an information string about why execution stopped and the current state of the machine.

load *file file file* ...
Read each of the given *files*. Treat them as DLX assembly language files and load memory as indicated in the files. Code (text) is normally loaded starting at address 0x100, but the **codeStart** variable may be used to set a different starting address. Data is normally loaded starting at address 0x1000, but a different starting address may be specified in the **dataStart** variable. The return value is either an empty string or an error message describing problems in reading the files. A list of directives that the loader understands is in a later section of this manual.

put *address number*
Store *number* in the register or memory location given by *address*. The return value is an empty string. To store floating point numbers (single or double precision), use the **fput** command.

quit Exit the simulator.

stats [**reset**] [**stalls**] [**opcount**] [**pending**] [**branch**] [**hw**] [**all**]
This command will dump various statistics collected by the simulator on the DLX code that has been run so far. Any combination of options may be selected. The options and their results are as follows:

reset Reset all of the statistics.

stalls Show the number of load stalls and stalls while waiting for a floating point unit to become available or for the result of a previous operation to become available.

opcount Show the number of each operation that has been executed.

pending Show all floating point operations currently being handled by the floating point units as well as what their results will be and where they will be stored.

branch Show the percentage of branches taken and not-taken.

hw Show the current hardware setup for the simulated machine.

all Equivalent to choosing all options except **reset**. This is the default.

step [*address*]
If no *address* is given, the **step** command executes a single instruction, continuing

from wherever execution previously stopped. If *address* is given, then the program counter is changed to point to *address*, and a single instruction is executed from there. In either case, the return value is an information string about the state of the machine after the single instruction has been executed.

stop [*option args*]

This command may take any of the forms described below:

stop Arrange for execution of DLX code to stop as soon as possible. If a simulation isn't in progress then this command has no effect. This command is most often used in the *command* argument for the **stop at** command. Returns an empty string.

stop at *address* [*command*]

Arrange for *command* (a **DLXsim** command string) to be executed whenever the memory address identified by *address* is read, written, or executed. If *command* is not given, it defaults to **stop**, so that execution stops whenever *address* is accessed. A stop applies to the entire word containing *address*: the stop will be triggered whenever any byte of the word is accessed. Stops are not processed during the **step** commands or the first instruction executed in a **go** command. Returns an empty string.

stop info

Return information about all stops currently set.

stop delete *number number number ...*

Delete each of the stops identified by the *number* arguments. Each *number* should be an identifying number for a stop, as printed by **stop info**. Returns an empty string.

ASSEMBLY FILE FORMAT

The assembler built into **DLXsim**, invoked using the **load** command, accepts standard format DLX assembly language programs. The file is expected to contain lines of the following form:

- Labels are defined by a group of non-blank characters starting with either a letter, an underscore, or a dollar sign, and followed immediately by a colon. They are associated with the next address to which code in the file will be stored. Labels can be accessed anywhere else within that file, and in files loaded after that if the label is declared as **.global** (see below).
- Comments are started with a semicolon, and continue to the end of the line.
- Constants can be entered either with or without a preceding number sign.
- The format of instructions and their operands are as shown in the Computer Architecture book.

While the assembler is processing an assembly file, the data and instructions it assembles are placed in memory based on either a text (code) or data pointer. Which pointer is used is selected not by the type of information, but by whether the most recent directive was **.data** or **.text**. The program initially loads into the text segment.

The assembler supports several directives which affect how it loads the DLX's memory. These should be entered in the place where you would normally place the instruction and its arguments. The directives currently supported by **DLXsim** are:

.align *n* Cause the next data/code loaded to be at the next higher address with the lower *n* bits zeroed (the next closest address greater than or equal to the current address that is a multiple of 2^{n-1}).

.ascii "*string1*", "*string2*", ...
: Store the *strings* listed on the line in memory as a list of characters. The strings are not terminated by a 0 byte.

.asciiz "*string1*", "*string2*", ...
: Similar to **.ascii**, except each string is followed by a 0 byte (like C strings).

.byte "*byte1*", "*byte2*", ...
: Store the *bytes* listed on the line sequentially in memory.

.data [*address*]
: Cause the following code and data to be stored in the data area. If an *address* was supplied, the data will be loaded starting at that address, otherwise, the last value for the data pointer will be used. If we were just reading code based on the text (code) pointer, store that address so that we can continue from there later (on a **.text** directive).

.double *number1*, *number2*, ...
: Store the *numbers* listed on the line sequentially in memory as double precision floating point numbers.

.float *number1*, *number2*, ...
: Store the *numbers* listed on the line sequentially in memory as single precision floating point numbers.

.global *label*
: Make the *label* available for reference by code found in files loaded after this file.

.space *size*
: Move the current storage pointer forward *size* bytes (to leave some empty space in memory).

.text [*address*]
: Cause the following code and data to be stored in the text (code) area. If an *address* was supplied, the data will be loaded starting at that address, otherwise, the last value for the text pointer will be used. If we were just reading data based on the data pointer, store that address so that we can continue from there later (on a **.data** directive).

.word *word1*, *word2*, ...

Store the *words* listed on the line sequentially in memory.

VARIABLES

DLXsim uses or sets the following Tcl variables:

codeStart

If this variable exists, it indicates where to start loading code in **load** commands.

dataStart

If this variable exists, it indicates where to start loading data in **load** commands.

insCount

DLXsim uses this variable to keep a running count of the total number of instructions that have been simulated so far.

prompt If this variable exists, it should contain a **DLXsim** command string. **DLXsim** will execute the command in this string before printing each prompt, and use the result as the prompt string to print. If this variable doesn't exist, or if an error occurs in executing its contents, then the prompt "(dlxsim)" is used.

SEE ALSO

Computer Architecture, A Quantitative Approach, by John L. Hennessy and David A. Patterson.

KEYWORDS

DLX, debug, simulate