

dlx user manual

Title	dlx (DLX functional model for ArchC)
Author	Nikolaos Kavvadias 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014
Contact	nikos@nkavvadias.com
Website	http://www.nkavvadias.com
Release Date	23 October 2014
Version	0.1.1
Rev. history	
v0.1.1	2014-10-23 <ul style="list-style-type: none">• Fixed wrong reference to memory MEM (correct is DM.• The simulator binary (<code>dlx.x</code>) is now correctly generated.
v0.1.0	2014-10-22 <ul style="list-style-type: none">• Updated documentation as README.rst.• Changed <code>func</code> from <code>0x06</code> to <code>0x09</code> for <code>multu</code>.• Removed <code>sequ</code>, <code>sneu</code>.• Changed <code>func</code> for <code>div</code> (<code>0x19</code>) and <code>divu</code> (<code>0x1A</code>).• Newly added files: <code>defines_gdb</code>, <code>modifiers</code>.• Updated <code>dlx_opcode_map.xls</code>.
v0.0.4	2006-11-15 <ul style="list-style-type: none">• Added <code>pcount</code> instruction for basic-block profiling.

v0.0.3	<p>2006-07-01</p> <ul style="list-style-type: none"> • Corrected optimization instruction methods for <code>j</code>, <code>jal</code>, <code>jr</code>, <code>jalr</code>, <code>beqz</code>, <code>bnez</code>. • Alternate behaviors for <code>div</code>, <code>divu</code> added. • Fixed copyright notations to manually-written files: • (<code>*.ac</code>, <code>dlx-isa.cpp</code>, <code>dlx_syscall.cpp</code>, <code>dlx_gdb_funcs.cpp</code>). • Behaviors for <code>addui</code>, <code>subui</code> have been corrected.
v0.0.2	<p>2006-01-01</p> <ul style="list-style-type: none"> • Changed behavior of <code>j</code>, <code>jal</code>, <code>beqz</code>, <code>bnez</code> according to what is expected by the binutils DLX port. • Fixed issue with <code>jr</code> instruction. • New encoding for the <code>halt</code> instruction. • Changed register notation to comply to DLX conventions: <code>(r0-r31)</code> and alternate notation: <code>(zero, at, v0-v1, a0-a3, t0-t9, s0-s7, k0-k1, gp, sp, fp, ra)</code> • Both prefixed (by a dollar sign) and unprefixed symbolic register names should be accepted. • Disabled non-standard DLX instructions, along with <code>mvts</code>, <code>mvfs</code>. • The standard <code>mult</code>, <code>multu</code>, <code>div</code>, <code>divu</code> opcodes are now used. • Testsuite directory removed. The <code>acstone</code> benchmarks should be used instead for the purpose of benchmarking the DLX model.

v0.0.1	2005-12-26 <ul style="list-style-type: none"> • First public version. • Most integer instruction set functionality has been added. • Very few applications have been tested: <ol style="list-style-type: none"> 1. <code>fib.s</code> (generated by <code>dlxgcc-2.7.2.3</code> and slightly modified) 2. <code>loadi.s</code> (tests load immediate pseudo-instructions)
---------------	---

1. Introduction

This is the DLX ArchC (<http://www.archc.org>) functional model. This model has the system call emulation functions implemented, so it is a good idea to turn on the ABI option.

To use `acsim`, the interpreted simulator:

```
$ acsim dlx.ac [-g -abi -gdb]      # (create the simulator)
$ make -f Makefile.archc          # (compile)
$ ./dlx.x --load=<file-path> [args] # (run an application)
```

To use `accsim`, the compiled simulator:

```
$ accsim dlx.ac <file-path>        # (create specialized simulator)
$ make -f Makefile.archc          # (compile)
$ ./dlx.x [args]                  # (run the application)
```

The `[args]` are optional arguments for the application.

There are two formats recognized for application `<file-path>`:

- ELF binary matching ArchC specifications
- hexadecimal text file for ArchC

To use `acasm`, the assembler generator:

```
$ asmggen.sh dlx.ac <assembler_name>
```

for generating the assembler source files.

Then, proceed to build it using the GNU autotools:

```
$ configure
$ make all-gas
$ make install-gas
```

2. File listing

The `dlx` distribution includes the following files:

<code>/dlx</code>	Top-level directory
<code>AUTHORS</code>	List of <code>dlx</code> authors.
<code>LICENSE</code>	The modified BSD license governs <code>dlx</code> .
<code>README.html</code>	HTML version of README.
<code>README.pdf</code>	PDF version of README.
<code>README.rst</code>	This file.
<code>VERSION</code>	Current version of the project sources.
<code>bp_conf.ac</code>	Branch predictor description (only for <code>archc-1.5.1.bp2</code>).
<code>defines_gdb</code>	Macro definitions for GDB integration.
<code>dlx.ac</code>	Register, memory and cache model for <code>dlx</code> .
<code>dlx_gdb_funcs.cpp</code>	GDB support for the DLX simulator.
<code>dlx_isa.ac</code>	Instruction encodings and assembly formats.
<code>dlx_opcode_map.vsd</code>	Incomplete MS Visio drawing of the DLX opcode map.
<code>dlx_opcode_map.xls</code>	Excel spreadsheet containing the DLX opcode map.
<code>dlx_syscall.cpp</code>	OS call emulation support for DLX.
<code>dlx-isa.cpp</code>	Instruction behaviors.
<code>modifiers</code>	Instruction encoding and decoding modifiers.
<code>rst2docs.sh</code>	Bash script for generating the HTML and PDF versions of the documentation (README).

3. General observations

1. Some non-classical DLX instructions (available in the DLX binutils target) might be added in the future. These are:
 - `bswap` (BSWAPF) --> A byte swap instruction
 - `ldstbu` (LSBUOP) --> Atomic load-store byte unsigned
 - `ldsthu` (LSHUOP) --> Atomic load-store halfword unsigned
 - `ldstw` (LSWOP) --> Atomic load-store word
2. `mult`, `multu`, `div`, `divu` instructions have different opcodes to the binutils DLX. Also, `div`, `divu` produce a single 32-bit result (the quotient). Probably, `rem`, `remu` instructions will be added to produce the remainder of a division. For 64-bit result multiplication maybe a good choice is to provide `multl`, `multlu` primitives, for which results are written in two consecutive registers (integer registers).
3. There are no `HI/LO` registers (I think this is the actual intent in the Patterson book).
4. Multiplication and division DONNOT use the floating-point register file. For this reason, `mvts`, `mvfs` instructions are currently unimplemented.

5. Loading 32-bit constants will be available via appropriate pseudo- instructions not requiring the HI/LO registers, and for the following formats:

```
li    %dest, #hi-16bit-constant, #lo-16bit-constant
li    %dest, #32bit-constant
```

6. For future provision of a coprocessor (maybe this is an overkill for the DLX?) some opcodes MIGHT be moved, e.g.:
- Move opcode(J)=0x02, opcode(JAL)=0x03 to e.g. 0x06,0x07, respectively. (PREFERRED)
 - Move opcode(BEQZ), opcode(BNEZ) to 0x16, 0x17.
 - Then the 0x01-0x04 primary opcodes would be used for 4 optional coprocessors.