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	22 October 2014
Version	0.1.0
Rev. history	
v0.1.0	2014-10-22 • Updated documentation as README.rst. • Changed `func` from `0x06` to `0x09` for `multu`. • Removed `sequ`, `sneu`. • Changed `func` for `div` (`0x19`) and `divu` `0x1A`. • Newly added files: `defines_gdb`, `modifiers`. • Updated `dlx_opcode_map.xls`.
v0.0.4	2006-11-15 • Added pcount instruction for basic-block profiling.
v0.0.3	2006-07-01 • Corrected optimization instruction methods for `j`, `jal`, `jr`, `jalr`, `beqz`, `bnez`. • Alternate behaviors for `div`, `divu` added. • Fixed copyright notations to manually-written files: • (`*.ac`, `dlx-isa.cpp`, `dlx_syscall.cpp`, `dlx_gdb_funcs.cpp`). • Behaviors for `addui`, `subui` have been corrected.

v0.0.2	2006-01-01 • Changed behavior of `j`, `jal`, `beqz`, `bnez` according to what is expected by the binutils DLX port. • Fixed issue with `jr` instruction. • New encoding for the `halt` instruction. • Changed register notation to comply to DLX conventions: (`r0-r31`) and alternate notation: (`zero,at,v0-v1,a0-a3,t0-t9,s0-s7, k0-k1,gp,sp,fp,ra`) • Both prefixed (by a dollar sign) and unprefixed symbolic register names should be accepted. • Disabled non-standard DLX instructions, along with `mvts`, `mvfs`. • The standard `mult`, `multu`, `div`, `divu` opcodes are now used. • Testsuite directory removed. The `acstone` benchmarks should be used instead for the purpose of benchmarking the DLX model.
v0.0.1	2005-12-26 • First public version. • Most integer instruction set functionality has been added. • Very few applications have been tested: 1. `fib.s` (generated by `dlxgcc-2.7.2.3` and slightly modified) 2. `loadi.s` (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:

```
$ asmgen.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:

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

modifiers	Instruction encoding and decoding modifiers.
rst2docs.sh	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.