# 372Project

LC2200 Simulator Manual

/\* TABLE OF CONTENTS

\*

\* littlecomputer2200 (LC2200) i.e., control unit

\* instructionsetarchitecture (bit)

\* systembus (bus)

\* finitestatemachine (fsm)

\* programcounter (pc)

\* arithmeticlogicunit (alu)

\* registerfile (reg)

\* randomaccessmemory (mem)

\* instructionregister (ir)

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* littlecomputer2200 (LC2200) i.e., control unit

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Functioning like the control unit, the LC2200 oversees all aspects of the

\* system and, like a conductor, tells them what to do by flipping signals in

\* each component. Everything the actual LC2200 computer contains--defined as very

\* modularized as simply the computer's logic circuits (struct), not the main,

\* the graphicaluserinterface (gui), or the assemblymachinereader (asm), or

\* anything else consisting or I/O--communicates through the LC2200 control unit

\* and not through each other.

\*

\* The control unit is responsible for setting the machine in its initial ready-

\* to-start stage. This consists of every possible value in the varous

\* components being set to 0 or equivalent (false, which in C is 0)--off.

\*

\* This is, in all respects, the brain of the machine. Each of the subcomponents

\* have mostly skeletal code: primitive types (see instructionsetarchitecture

\* (bit) for more info), and very short functions that simply update the struct

\* values from the bus or vice versa. This is the handshake that the machine

\* makes with its parts: you are to do nothing unless instructed--and then

\* you are only responsible for exchanging data. The only component that

\* vary slightly from this is the alu, although it is simply comprised on a few

\* very short functions following the handshake rules as well.

\*/

/\* SIGNAL ENUM

\* -----------------------------------------------------------------------------

\* Used in the three macro-stage cycle to emulate drive, load, and write.

\*/

typedef enum {

Dr, Ld, Wr

} signalx;

/\* COMPONENT ENUM

\* -----------------------------------------------------------------------------

\* The current component that the control unit is sending signals to.

\*/

typedef enum {

\_pc, \_aluA, \_aluB, \_reg, \_mem, \_ir

} componentx;

/\* littlecomputer (LC2200) struct

\* -----------------------------------------------------------------------------

\* The setup of the LC2200, including all of the components and several bit

\* switches for operations and debugging.

\*/

typedef struct {

bit safetydebug;

bit statedebug;

bit microdebug;

bit clock;

bit z;

word cycle;

fsm\_ fsm;

pc\_ pc;

alu\_ alu;

reg\_ reg;

mem\_ mem;

ir\_ ir;

} littlecomputer2200;

/\* POINTER \*/

typedef littlecomputer2200 \*LC2200\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

LC2200\_ LC2200\_ctor();

void LC2200\_kill(LC2200\_);

/\* METHODS \*/

void start(LC2200\_, char); //starts the system under certain conditions

void setupcycle(LC2200\_); //cycle phase i

void switchstate(LC2200\_); //cycle phase ii

void microstate(LC2200\_); //cycle phase iii

void calleesave(LC2200\_); //unimplemented

void debug(LC2200\_); //for testing

/\* KILL

\* -----------------------------------------------------------------------------

\* Kills every one of its components and then falls on the sword itself.

\*/

/\* START

\* -----------------------------------------------------------------------------

\* Starts the system. Upon start, the machine will run until it reaches a halt,

\* hits a predefined safety limit, a certain amount of steps, or a jumping or

\* branching instruction.

\*/

/\* CYCLE PHASE I

\* -----------------------------------------------------------------------------

\* Reads the next state in the finitestatemachine (fsm) and flips every signal

\* in the computer's components to what it will need to be for the next

\* microstate. This ensures that when the load, drive, and write signals are

\* fired, the appropriate parts of the system will be primed to write to and

\* read from the bus.

\*/

/\* CYCLE PHASE II

\* -----------------------------------------------------------------------------

\* Depending on the last four bits of the ROM state, the control unit will

\* a series of elaborate check to determine what the next microstate will be.

\* The factors effecting its decision are the instructionregister (ir)'s opcode

\* value, the current state's 10 bits of state information, the z-value attached

\* to the value of the bus, and additional constraints on the micro cycle. See

\* the finitestatemachine (fsm) for more information.

\*/

/\* CYCLE PHASE III

\* -----------------------------------------------------------------------------

\* For each stage in the drive-load-write progression, sends out an activation

\* signal to each component. If that component's respective signal is flipped

\* true, then the component performs its indicated action, either pushing a word

\* onto the bus or reading from the bus and storing or writing the information.

\*/

/\* DEBUG OUTPUT

\* -----------------------------------------------------------------------------

\* Prints to console the state of every struct data piece of every component.

\* Will omit finitestatemachine (fsm)'s ROM, registerfile (reg)'s reserved and

\* callee-save bits, and randomaccessmemory (mem)'s MEM to a certain limit

\* based on the LC2200's header file.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* instructionsetarchitecture (bit)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The bit file contains definitions for the actual data types underlying the

\* LC2200 and all of its components and accessories. The reason for typedef-ing

\* a data type was so that it could be changed relatively quickly if was needed

\* --by having all of its data types coded this way (and having all of the

\* conversion of data types be handled through the .c version of bit), we were

\* enabled to try several formats. Ultimately, it was decided that:

\*

\* data type internal underlying purpose

\* boolean bit \_Bool simple 2-value switch, or bit

\* 32-bit long word uint32\_t 4 byte values for registers, memory

\* signed long aluw int32\_t special signed 4 byte word for alu

\* 32-bit long bus word connecting the system dialogues

\*/

/\* SYSTEM WIDE CASTS DEFINITIONS \*/

typedef \_Bool bit; //boolean signal/switch/bit

typedef uint32\_t word; //four bytes, system basis

typedef int32\_t aluw; //special signed word for the alu

/\* GLOBAL DECLARATION OF BUS \*/

extern word bus;

/\* BIT, WORD, STRING, and SPECIAL CONVERSION METHODS \*/

word bits(word, int, int); //returns subarray of word as word

bit bitt(word, int); //returns single bit in word as bit

word stow(char \*); //returns binary string as word

char \*wtos(word); //returns word as binary string

word sigx(word, bit); //returns sign-extended word at given bit on

/\* EXTRACT BITS

\* -----------------------------------------------------------------------------

\* Given a word and a starting and ending bit, extracts the bits from start to

\* end inclusive and converts them to a word (least-significant bit aligned)

\* and returns that word.

\*/

/\* EXTRACT BIT

\* -----------------------------------------------------------------------------

\* Given a word and bit position, extracts the boolean value from that bit and

\* returns it as a bit.

\*/

/\* CONVERT STRING TO WORD

\* -----------------------------------------------------------------------------

\* Given a binary string (with any number of other characters), extracts the 1s

\* and 0s in sequential order and converts them to a word (least-significant bit

\* aligned) which is returned.

\*/

/\* CONVERT WORD TO STRING

\* -----------------------------------------------------------------------------

\* Given a word, converts it into a binary string representation of 1s and 0s.

\*/

/\* SIGN EXTENSION

\* -----------------------------------------------------------------------------

\* Given a word and a final bit position, extends the value of the final bit

\* towards the most-significant bit, filling in all bits with that value and

\* returning the resulting word.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* systembus (bus)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The bus acts as the system bus in the LC2200 (see LC2200 for control usage).

\* The bus can be read from and written to by all components, and is used in

\* this manner to simulate the transportation of information in 32 wires (one

\* word, as defined in bit's header file) when triggered by the control unit.

\* Almost every microstage involves the driving of a word onto the bus and the

\* reading and/or storage of that word elsewhere.

\*/

/\* GLOBAL DECLARATION OF BUS

\* -----------------------------------------------------------------------------

\* The bus acts as the system bus in the LC2200, responsible for carrying almost

\* all of its data. It is an external variable as declared by the

\* instructionsetarchitecture (bit)'s header file. With this usage, it is

\* immediate accessible to all components of the LC2200. It can be read from or

\* written to by any part of the system when signaled by the control unit (the

\* LC2200). See bit (.h) for more info.

\*

\* The bus/control-unit/component communicaiton is only broken for two reasons.

\* The LC2200 control unit can interact with the instructionregister (ir) and

\* the arithmeticlogicunit (alu) directly for purposes of determining the next

\* microstate to advance to using bits 28-31 of the finitestatemachine (fsm)'s

\* current state.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* finitestatemachine (fsm)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The finite state machine, containing an array of all possible microstates the

\* computer can be in. Apart from containing the ROM of microstates, relies on

\* external control by the LC2200 to progress to the next finite state. Used for

\* state transition by flipping the appropriate load, drive, and write switches

\* and updating the state to be next executed. The fsm is organized with the

\* first 10 bits representing the current state's opcode, z-value, and state--

\* these are used for finding the next appropriate state later. The next 13 bits

\* tell the control unit what drive, load, and write signals should be on or

\* off. The next 5 bits tell the control unit what function the alu should

\* execute and which of the three registers the ir should drive.

\*

\* Layout of FSM by line:

\* sig opcode z-value state drive load write alufunc irreg nextstate

\* com - - - PARMO PABMIZ MR - - -

\* idx 0123 4 56789 01234 567890 12 345 67 8901

\* bin 0000 0 00000 00000 000000 00 000 00 0000

\*

\* For bit flipping and signaling, when updating, the fsm will provide bit

\* switches to explain how each current state should determine the next

\* sequential state by using the last four bits in the microstate line. It is

\* expected that the control unit take care of the following conditions: if the

\* current state is ifetch3, the bit sequence 1000 will search only opcodes

\* using the opcode in the ir; if the current state is beq3, the control unit

\* reads primes the z-value by triggering the alu. The default starting state

\* upon boot-up is at ifetch1.

\*/

/\* finitestatemachine (fsm) struct

\* -----------------------------------------------------------------------------

\* Contains a word state, which is where in the ROM the current microstate is,

\* and the ROM itself, which provides for the control unit information on what

\* signals to send out for every microstate.

\*/

typedef struct {

word state;

word ROM[ROM\_SIZE];

} finitestatemachine;

/\* POINTER \*/

typedef finitestatemachine \*fsm\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

fsm\_ fsm\_ctor();

void fsm\_kill(fsm\_);

/\* MICROSTATE BIT DETAIL

\*

\* MICROSTATE ROM LAYOUT

\* op z st dr ld wr fn rg +

\* code - ##### PARMO PABMIZ MR ### ## OZ+S

\* 0123 4 56789 01234 567890 12 345 67 8901

\*

\* op 0123 the opcode address of the command

\* z 4 the z address of the command

\* st 56789 the state address of the command

\* dr 01234 drive bits in order: PC,ALU,REG,MEM,OFF

\* ld 567890 load bits in order: PC,ALU-A,ALU-B,MAR,IR,Z

\* wr 12 write bits in order: MEM,REG

\* fn 345 function bits to ALU

\* rg 67 register bits to IR

\* + 8901 how to go to the next step:

\* O 27: use opcode in address

\* Z 28: use z-val in address

\* + 29: increment and use state in address

\* S 30: whether to invoke callee-save

\*

\*/

/\* KILL

\* -----------------------------------------------------------------------------

\* Frees and nullifies the fsm.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* programcounter (pc)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Holds the value of the next instruction to be processed--if its drive or load

\* signal is enable, can push or replace this value. Used by the LC2200 as a

\* marker indicated where in the code it currently is and where to go next.

\*/

/\* programcounter (pc) struct

\* -----------------------------------------------------------------------------

\* Contains the drive and load bits as well as a register for holding the

\* current program counter.

\*/

typedef struct {

bit DrPC;

bit LdPC;

word pc;

} programcounter;

/\* POINTER \*/

typedef programcounter \*pc\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

pc\_ pc\_ctor();

void pc\_kill(pc\_);

/\* BUS METHODS \*/

void pc\_Dr(pc\_);

void pc\_Ld(pc\_);

/\* KILL

\* -----------------------------------------------------------------------------

\* Frees and nullifies the pc.

\*/

/\* DRIVE

\* -----------------------------------------------------------------------------

\* If the drive signal is enabled, pushes the current pc onto the bus.

\*/

/\* LOAD

\* -----------------------------------------------------------------------------

\* If the load signal is enabled, replaces the current pc with the bus value.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* arithmeticlogicunit (alu)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Responsible for performing all of the mathematical operations in the entire

\* computer. The alu contains two dedicated special purpose signed word

\* registers for use in these calculations and a three-bit function multiplexor

\* for determining which calculation to do.

\*/

/\* arithmeticlogicunit (alu) struct

\* -----------------------------------------------------------------------------

\* Contains the drive and both load bits (one for register A and B) as well as

\* two special signed registers for holding the operands, A and B, and a bit

\* switch array for determining what mathematical operator to use on A and B.

\* current program counter.

\*/

typedef struct {

bit DrALU;

bit LdA;

bit LdB;

aluw A;

aluw B;

bit func[NUM\_FUNC];

} arithmeticlogicunit;

/\* POINTER \*/

typedef arithmeticlogicunit \*alu\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

alu\_ alu\_ctor();

void alu\_kill(alu\_);

/\* BUS METHODS \*/

void alu\_Dr(alu\_);

void alu\_LdA(alu\_);

void alu\_LdB(alu\_);

/\* SPECIAL METHODS \*/

//note: last four functions unimplemented in fsm

void alu\_addb(alu\_); //000 a + b

void alu\_nand(alu\_); //001 a nand b

void alu\_asub(alu\_); //010 a - b

void alu\_ainc(alu\_); //011 a + 1

void alu\_aorb(alu\_); //100 a or b

void alu\_andb(alu\_); //101 a and b

void alu\_axrb(alu\_); //110 a xor b

void alu\_nota(alu\_); //111 not a

/\* KILL

\* -----------------------------------------------------------------------------

\* Frees and nullifies the alu.

\*/

/\* DRIVE

\* -----------------------------------------------------------------------------

\* If the drive signal is enabled, performs the operation indicated by regno

\* and pushes the result to the bus.

\*/

/\* LOAD A

\* -----------------------------------------------------------------------------

\* If the load A signal is enabled, updates the A register with the bus value.

\*/

/\* LOAD B

\* -----------------------------------------------------------------------------

\* If the load B signal is enabled, updates the B register with the bus value.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* registerfile (reg)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The heart of the LC2200 is the registerfile, a group of 16 registers that

\* are the only components the cpu can interact with. Each of the registers has

\* a specific index, name, and write and read controls. The parallel arrays

\* of reserved bits and callee-save bits indicate which registers are

\* write-protected and save-protected respectively. The index of the current

\* register activated is determined by a regno multiplexor.

\*/

/\* SYSTEM WIDE DECLARATION OF REGISTER NAMES \*/

const char \*REG\_NAMES[] = {

0 1 2 3 4 5 6 7

"$zero","$at", "$v0", "$a0", "$a1", "$a2", "$t0", "$t1",

8 9 10 11 12 13 14 15

"$t2", "$s0", "$s1", "$s2", "$k0", "$sp", "$fp", "$ra"

};

\* registerfile (reg) struct

\* -----------------------------------------------------------------------------

\* Contains the drive and write bits as well as a register for holding the

\* the currently selected register, two parallel arrays of bit switches for

\* determining writability and callee-save status, which are parallel to the

\* registers themselves, REG.

\*/

typedef struct {

bit DrREG;

bit WrREG;

word regno;

bit reserved[REG\_NUM];

bit calleesave[REG\_NUM];

word REG[REG\_NUM];

} registerfile;

/\* POINTER \*/

typedef registerfile \*reg\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

reg\_ reg\_ctor();

void reg\_kill(reg\_);

/\* BUS METHODS \*/

void reg\_Dr(reg\_);

void reg\_Wr(reg\_);

/\* SPECIAL METHODS \*/

//note: both methods unimplemented

void reg\_writeReserved(reg\_);

word save(reg\_);

/\* KILL

\* -----------------------------------------------------------------------------

\* Frees and nullifies the reg.

\*/

/\* DRIVE

\* -----------------------------------------------------------------------------

\* If the drive signal is enabled, the bus is updated to contain the register

\* value indicated by regno.

\*/

/\* WRITE

\* -----------------------------------------------------------------------------

\* If the write signal is enabled and the respective reserved array indicates

\* the register at regno is write-enabled, updates that regsiter with the bus.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* randomaccessmemory (mem)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* The memory is the largest component of the LC2200, acting as the storage

\* behind the entire system. Although the book's LC2200 contained 2^32 memory

\* addresses, this would constitute > 17gb worth of data--far too large for

\* a regular computer to run efficiently. Thus, the actual size of memory is

\* determined in the mem header file and runs from a minimum of 2^20 memory

\* addresses (a little over a million), which shouldn't take up more than a

\* little over 4mb of actual ram size, to a reliable maximum of 2^28 memory

\* addresses (about 256 million), which would take up over 1gb of ram size.

\*

\* The mem has drive, load, and write components. All three utilized the value

\* stored in the memory address register (MAR) to choose which memory cell to

\* interact with the bus.

\*/

/\* randomaccessmemory (mem) struct

\* -----------------------------------------------------------------------------

\* Contains the drive, load, and write bits, the memory address register (MAR)

\* which is used for address lookup, and the memory itself (MEM), which is the

\* actual memory contents.

\*/

typedef struct {

bit DrMEM;

bit LdMAR;

bit WrMEM;

word MAR;

word MEM[]; //unspecified array length so compiler uses heap and not stack

} randomaccessmemory;

/\* POINTER \*/

typedef randomaccessmemory \*mem\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

mem\_ mem\_ctor();

void mem\_kill(mem\_);

/\* BUS METHODS \*/

void mem\_Dr(mem\_);

void mem\_Ld(mem\_);

void mem\_Wr(mem\_);

/\* KILL

\* -----------------------------------------------------------------------------

\* Frees and nullifies the mem.

\*/

/\* DRIVE

\* -----------------------------------------------------------------------------

\* If the drive signal is enabled, pushes MEM[MAR] onto the bus.

\*/

/\* LOAD

\* -----------------------------------------------------------------------------

\* If the load signal is enabled, sets MAR to bus contents.

\*/

/\* WRITE

\* -----------------------------------------------------------------------------

\* If the write signal is enabled, writes to MEM[MAR] from the bus.

\*/

/\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* instructionregister (ir)

\* \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Contains the register for holding the current instruction and drive and load

\* bits. The LC2200 uses the ir for determining what opcode to set the fsm to,

\* pulling the registers need for calculations, and for sign-extended the

\*/

/\* instructionregister (ir) struct

\* -----------------------------------------------------------------------------

\* Contains the drive and load bits as well as a register for holding the

\* current instruction.

\*/

typedef struct {

bit DrOFF;

bit LdIR;

word instruction;

} instructionregister;

/\* POINTER \*/

typedef instructionregister \*ir\_;

/\* CONSTRUCTOR/DESTRUCTOR \*/

ir\_ ir\_ctor();

void ir\_kill(ir\_);

/\* BUS METHODS \*/

void ir\_Dr(ir\_);

void ir\_Ld(ir\_);

/\* SPECIAL METHODS \*/

word ir\_opc(ir\_);

word ir\_reg(ir\_, word);

/\* KILL

\* -----------------------------------------------------------------------------

\* Frees and nullifies the ir.

\*/

/\* DRIVE

\* -----------------------------------------------------------------------------

\* If the drive signal is enabled, sign-extends the last 20 bits of the

\* instruction and pushes them onto the bus.

\*/

/\* LOAD

\* -----------------------------------------------------------------------------

\* If the load signal is enabled, updates the register with bus value.

\*/

/\* OPCODE RETRIEVE

\* -----------------------------------------------------------------------------

\* Returns as a word the opcode (first 4 bits) of the current instruction.

\*/

/\* REGISTER RETRIEVE

\* -----------------------------------------------------------------------------

\* Given the positional register desired, returns as a word that register's

\* number. Each possible register storage space in the instruction is four bits

\* long, or 2^4, the exact amount of registers there are in the registerfile

\* (reg). Setup as follows:

\*

\* reg OPCODE REGX/1 REGY/2 IMMEDIATE/UNUSED REGZ/3/IMMEDIATE/UNUSED

\* bit 0123 4567 8901 2345678901234567 8901

\*/