LEO-1 Homebrew Computer

Registers, Control Signals and Buses

Rev 1.5, 21st June 2017, John Croudy

Introduction

The LEO-1 CPU, like most digital computers, operates by fetching, decoding and executing *instructions* which are stored in memory. Each *instruction cycle* results in the execution of one instruction. These instructions perform operations on registers and sometimes memory or memory-mapped devices. The Control Unit coordinates this process using a kind of state machine built from combinational logic. The whole thing is run by a system clock which is essentially just an oscillator. Each tick of this oscillator is called a *clock cycle*.

Fetching and execution of instructions proceeds in up to eight *states*, each state corresponding to one clock cycle. Each clock tick moves to the next state. The first state is called *State 0* and the last is called *State 7*. The period of State 0 to 3 is called *Phase 1* and the period of State 4 to 7 is called *Phase 2*.

This document describes the electrical signals that LEO-1 uses while it is running and what these signals do. There are individual signals that originate in the state machine and groups of signals (*buses*) that originate in registers.

Registers

In addition to the eight general-purpose registers in the Register Unit, there are several Control Unit registers which are not directly visible to the programmer. These registers are listed in the table below.

Name	Width	Function
PC	16	The *Program Counter* contains the address of the instruction being fetched and executed. At the end of the instruction cycle, it is usually incremented, but during a branch instruction a value between -128 and 127 is added to it. During the instruction fetch, the I-Bank and PC are together placed on the address bus to form the 24-bit address of the instruction to be fetched.
IR	16	The *Instruction Register* holds the current instruction. During states 0 and 1 this data is invalid. At state 2, the instruction fetch occurs and by state 3 the IR is stable and valid. This register creates the signals `IR0` to `IR15`.
I-Bank	8	The *Instruction Bank* register specifies the high 8 bits of the address of the instruction to fetch. During the instruction fetch, this value is placed on bits 23 to 16 of the address bus while the PC is placed on bits 15 to 0 of the address bus.
D-Bank	8	The *Data Bank* register specifies the high 8 bits of the address of a memory load or store operation. During a memory access instruction, this value is placed on bits 23 to 16 of the address bus while the held R Bus is placed on bits 15 to 0 of the address bus.

Signals

In the table below, the logic for each signal is defined in terms of other signals. This is written in the same way that logic operations are specified in the C language. The signals are all derived from either state signals or bits from the instruction register. These signal names appear in the schematics and also in the Logisim simulation.

Signals with names that start with / use negative logic (active low). All other signals use positive logic (active high). Because 74 series chips tend to use active low enable signals, the majority of the single-bit signals in LEO-1 use negative logic. All multi-bit buses and registers, however, use positive logic.

When a signal is referred to as being 'active' or 'asserted', both terms mean the signal's logic state is *true*. Conversely, 'inactive' and 'unasserted' mean *false*. For positive logic signals, *true* means the associated wire or PCB trace is high (5 volts). For negative logic signals, *true* means the associated wire or PCB trace is low (0 volts).

Name	Pg	Description
/STATEn	C3	There can be up to eight states in each instruction cycle, each state lasting for one clock cycle. Exactly one /STATEn signal is active during each state.
IRn	C7	There are 16 bits in one instruction word. After an instruction has been fetched into the instruction register on state 2, each instruction bit creates a signal called IR0 to IR15. On any given cycle, these bits are invalid prior to state 2 as they will still contain the data from the previous instruction.
/PHASE1	C3	This signal is asserted during phase 1 of the instruction cycle. This occurs during states 0 to 3. /PHASE1 = /STATE0 || /STATE1 || /STATE2 || /STATE3
/PHASE2	C3	This signal is asserted during phase 2 of the instruction cycle. This occurs during states 4 to 7. /PHASE2 = /STATE4 || /STATE5 || /STATE6 || /STATE7
/RESET	C3	This signal is asserted when the reset button is pressed. It is connected to all registers and the state counter which it clears to zero.

/OPREG	C7	This signal indicates that the instruction is Type 0 meaning a register or memory instruction. `/OPREG = !IR15 && !IR14`
/OPIMM	C7	This signal indicates that the instruction is Type 1 meaning an immediate instruction. `/OPIMM = !IR15 && IR14`
/OPJPBR	C7	This signal indicates that the instruction is Type 2 meaning a jump or branch instruction. `/OPJPBR = IR15 && !IR14`
/OPMISC	C7	This signal indicates that the instruction is Type 3 meaning a miscellaneous instruction. `/OPMISC = IR15 && IR14`
/SWHL	C4	This signal is asserted when the instruction is the **swhl** instruction. `/SWHL = /OPMISC && IR0 && IR1 && !IR2`
/REGIMM	C7	This signal is asserted when `/OPREG` is asserted or `/OPIMM` is asserted. `/REGIMM = /OPREG || /OPIMM`
/REGIMEX	C7	This signal is asserted when `/REGIMM` is asserted, or `/SWHL` is asserted. `/REGIMEX = /REGIMM || /SWHL`
/OPT	C3	This signal indicates that the instruction cycle is being optimized for speed. A full eight clock cycles is only needed to complete memory load and store operations. All other operations can be completed in only five cycles. Therefore, when the state counter reaches state 4, if `/MEM` is not asserted, `/OPT` is asserted. This causes the `/DATWRI` (and sometimes `/REGWRI`) signal to be asserted early, which completes the necessary register write for that instruction. If `/OPT` is active at state 5, the state counter resets to zero and the instruction ends early. This optimization can be switched off by using a jumper. This is an experiment to see what affect it has on the overall speed of a program. `/OPT = (/STATE4 || /STATE5) && !/MEM`

/IRCODEn	C4	This is a set of internal signals that are generated by the low three bits of the instruction register. They are used by miscellaneous and jump/branch instructions.
		
		`/IRCODE0 = !IR2 && !IR1 && !IR0`
		`/IRCODE1 = !IR2 && !IR1 &&  IR0`
		`/IRCODE2 = !IR2 &&  IR1 && !IR0`
		`/IRCODE3 = !IR2 &&  IR1 &&  IR0`
		`/IRCODE4 =  IR2 && !IR1 && !IR0`
		`/IRCODE5 =  IR2 && !IR1 &&  IR0`
		`/IRCODE6 =  IR2 &&  IR1 && !IR0`
		`/IRCODE7 =  IR2 &&  IR1 &&  IR0`

/JUMP	C4	This signal is asserted when the instruction is the **jump** instruction. `/JUMP = /OPJPBR && /IRCODE0`
/HALT	C4	This signal is asserted when the instruction is the **halt** instruction. `/HALT = /OPMISC && /IRCODE7`
/CR	C4	When active, this signal resets the state counter to zero. It is asserted when `/HALT` is active. It is also asserted if `/OPT` is active on state 5 and thus ends the instruction early. `/CR = /HALT || (/OPT && /STATE5)`
/DATWRI	C3	This is an intermediate signal with several purposes. It is used to generate the `/BANKSTB`, `PCWRI` and `/REGWRI` signals. It is asserted during state 6 (or state 4 if `/OPT` is asserted). `/DATWRI = /STATE6 || /OPT`
/REGWRI	C3	This signal causes the Register Unit to write the contents of the `RIN` bus to the C-selected register. `/REGWRI = /REGIMEX && /DATWRI && !/STORE`
/BANK	C4	This is an intermediate signal which indicates that the instruction is a **bank** or **banki** instruction. `/BANK = /OPMISC && (/IRCODE1 || /IRCODE2)`
/BANKWW	C4	This is an intermediate signal which indicates that the instruction is a **jump**, **bank** or **banki** instruction, all of which require a write to a bank register. `/BANKWW = /JUMP || /BANK`

/BANKWRI	C4	This signal is the bank write strobe. When asserted, one of the two bank registers gets a write signal and stores the data currently on HRBUS<7:0>. Which bank register performs the write depends on other signals. /BANKWRI = /BANKWW && /DATWRI
/BREGLIT	C4	This signal indicates that the instruction is an immediate instruction, either immediate to register or immediate to bank. It is used to decide whether to use the B Bus or the literal value instruction bits as the ALU 'B' operand. /BREGLIT = /OPIMM || (/OPMISC && /IRCODE1)
/ZERO	C4	This signal is asserted when the data on the held C Bus is zero. /ZERO = (HCBUS == 0)
/PLUS	C4	This signal is asserted when the signed value on the held C Bus is positive. /PLUS = !HCBUS<15>
/TAKEBRA	C4	This signal is asserted when the instruction is a branch instruction and the branch will be taken. CANBRA = /IRCODE7 || (/IRCODE3 && /ZERO) || (/IRCODE4 && !/ZERO) || (/IRCODE5 && /PLUS) || (/IRCODE6 && !/PLUS) /TAKEBRA = /OPJPBR && CANBRA
IMEM	C7	This is an alias for IR3. When the instruction is Type 0 it indicates a memory read or write operation. IMEM = IR3
IMEMWR	C7	This is an alias for IR4. When the instruction is Type 0 and IMEM is asserted, it indicates a memory write operation. IMEMWR = IR4
/MEM	C3	This signal is asserted when the instruction is a memory instruction. /MEM = /OPREG && IMEM
/MEMLD	C3	This signal is asserted when the instruction is a memory load instruction. /MEMLD = /MEM && !IMEMWR

/STORE	C3	This is an intermediate signal used to generate several other signals. It is asserted when the instruction is a memory store instruction. `/STORE = /MEM && IMEMWR`
/MEMW	C3	This signal is asserted during states 4, 5 and 6 when the instruction is a memory store instruction. `/MEMW = (/STATE4 || /STATE5 || /STATE6) && /STORE`
/MEMEN	C3	This signal is the select enable for memory and devices. When this signal is asserted, the memory unit's address decoder enables the device that is assigned to the address currently on the address bus. This device may then perform a read (if `/OE` is asserted) or a write (if `/WE` is asserted). `/MEMEN` is always asserted during states 1 and 2 so that the current instruction can be fetched. It is also asserted during states 5 and 6 if the instruction is a memory instruction. `/MEMEN = /STATE1 || /STATE2 || (/MEM && (/STATE5 || /STATE6))`
/IRCLK	C3	This signal is asserted during state 2. It causes the data on the held Data Bus to be latched into the instruction register.
/OE	C3	This signal is the output enable for memory and devices. When active, the Memory Board allows the currently addressed device to place data on the Data Bus. It is always asserted during states 0 to 2 so that the current instruction can be fetched. From state 3 until the end of the cycle, it is only asserted when the instruction is not a memory write. `/OE = /STATE0 || /STATE1 || /STATE2 || !/STORE`
/WE	C3	This signal is connected to the write-enable input of all RAM and other devices that can be written to on the Memory Unit. During state 5 it provides the write strobe that causes the device to store the contents of the data bus. `/WE = /STATE5 && /STORE` **Timing note:** Although this signal is asserted at the same time as `/MEMEN`, it is deliberately delayed by about 50 ns by exploiting the propagation delay of six inverters in series. The reason is that `/MEMEN` would otherwise be asserted a few nanoseconds *after* `/WE` because there is an extra gate in the logic path that generates it. While this wouldn't matter for the memory chips the design uses (which write transparently), if we were to later add devices that are edge-sensitive on `/WE`, they might not work if `/WE` was asserted slightly before `/MEMEN`. By delaying `/WE`, we ensure that `/MEMEN` comes first and `/WE` goes on and off while `/MEMEN` is still asserted.

/EWE	C3	This is the original /WE signal before it is delayed (see /WE timing note).
/REGSE	C3	This signal is the select enable for registers. When active, the Register Unit allows registers to place data on the A, B and C Buses. It also allows a register to be written to when /REGWRI is asserted. /REGSE = /STATE3 || /PHASE2
/ALUOE	C3	When this signal is asserted, the ALU output is enabled onto the R-Bus. Since the ALU must be enabled at the same time as the registers, /ALUOE is the same signal as /REGSE. /ALUOE = /REGSE
/PCREG	C3	This signal indicates that a register instruction is in fact a PC-to-register instruction. It is asserted whenever the instruction is a register instruction and the instruction has bit 4 active and bit 3 inactive. /PCREG = /OPREG && IR4 && !IR3
/EFFADR	C3	This signal is asserted during phase 2 when the instruction is a memory instruction. /EFFADR = /PHASE2 && /MEM
IBANKMUX	C7	This is an alias for IR5 which is used to generate ALUB. /IBANKMUX = /IR5
IALU0	C7	This is an alias for IR0. It is bit 0 of a 4-bit code in instructions that specifies the ALU operation. /IALU0 = /IR0
IALU1	C7	This is an alias for IR1. It is bit 1 of a 4-bit code in instructions that specifies the ALU operation. /IALU1 = /IR1
IALU2	C7	This is an alias for IR2. It is bit 2 of a 4-bit code in instructions that specifies the ALU operation. /IALU2 = /IR2
IALU3	C7	This is an alias for IR15. It is bit 3 of a 4-bit code in instructions that specifies the ALU operation.

		/IALU3 = /IR15
IREGSA0	C7	This is an alias for IR8. It is bit 0 of a 3-bit code in register instructions that specifies the A register. /IREGSA0 = /IR8
IREGSA1	C7	This is an alias for IR9. It is bit 1 of a 3-bit code in register instructions that specifies the A register. /IREGSA1 = /IR9
IREGSA2	C7	This is an alias for IR10. It is bit 2 of a 3-bit code in register instructions that specifies the A register. /IREGSA2 = /IR10
IREGSB0	C7	This is an alias for IR5. It is bit 0 of a 3-bit code in register instructions that specifies the B register. /IREGSB0 = /IR5
IREGSB1	C7	This is an alias for IR6. It is bit 1 of a 3-bit code in register instructions that specifies the B register. /IREGSB1 = /IR6
IREGSB2	C7	This is an alias for IR7. It is bit 2 of a 3-bit code in register instructions that specifies the B register. /IREGSB2 = /IR7
IREGSC0	C7	This is an alias for IR11. It is bit 0 of a 3-bit code in register instructions that specifies the C register. /IREGSC0 = /IR11
IREGSC1	C7	This is an alias for IR12. It is bit 1 of a 3-bit code in register instructions that specifies the C register. /IREGSC1 = /IR12
IREGSC2	C7	This is an alias for IR13. It is bit 2 of a 3-bit code in register instructions that specifies the C register. /IREGSC2 = /IR13

Buses

Name	Width	Function
DATABUS	16	This is a bidirectional bus which allows transfer of data to or from a memory-mapped device. When no device is enabled, this bus goes open-circuit. Therefore, inside the Control Unit, the last state of this bus is held by a bus-hold circuit, creating an internal 'held Data Bus': *HDATABUS*. `if (/MEMW) DATABUS = HCBUS`

ADDRBUS	24	Specifies the address of a device from which data is to be read or to which data is to be written.
		
		`ADDRBUS<15:0>  =  /EFFADR             ? HRBUS  : PC`
		`ADDRBUS<23:16> = (/EFFADR && !/PCREG) ? D-Bank : I-Bank`

RIN	16	This bus is connected to the inputs of each register on the Register Unit. It carries the data to be written during a register write operation. `RIN = /MEMLD ? HDATABUS : HRBUS`
ABUS	16	Connected between the Register Unit and the A input of the ALU, this specifies the ALU 'A' operand. When no register is selected, this bus goes open-circuit. Therefore, inside the ALU, the last state of this bus is held by a bus-hold circuit, creating an internal 'held A bus': *HABUS*.
BBUS	16	Connected between the Register Unit and the Control Unit, this specifies the ALU 'B' operand when the instruction is a register instruction. When no device is enabled, this bus goes open-circuit. Therefore, inside the Control Unit, the last state of this bus is held by a bus-hold circuit, creating an internal 'held B Bus': *HBBUS*.
CBUS	16	Connected between the Register Unit and the Control Unit, this carries the data to be written during a register to memory write operation. It is also this data that is checked when deciding whether or not to take a conditional branch. When no register is selected, this bus goes open-circuit. Therefore, inside the Control Unit, the last state of this bus is held by a bus-hold circuit, creating an internal 'held C bus': *HCBUS*.

ALUB	16	Connected between the Control Unit and the B input of the ALU, this specifies the ALU 'B' operand.
		
		`P = IBANKMUX ? BANKOUT : PCVAL`
		`Q = /BREGLIT ? ILITVAL : HBBUS`
		`ALUB = /PCREG ? P : Q`

RBUS	16	This bus carries the result of an ALU operation from the ALU to the Control Unit. When the ALU output is disabled, this bus goes open-circuit. Therefore, inside the Control Unit, the last state of this bus is held by a bus-hold circuit, creating an internal 'held R bus': *HRBUS*.

REGSAn	3	This is the A-register select control bus. It is used by the Control Unit to tell the Register Unit which register to select onto the A bus.
		
		`REGSA0 = /OPIMM ? IREGSC0 : IREGSA0`
		`REGSA1 = /OPIMM ? IREGSC1 : IREGSA1`
		`REGSA2 = /OPIMM ? IREGSC2 : IREGSA2`

REGSBn	3	This is the B-register select control bus. It is used by the Control Unit to tell the Register Unit which register to select onto the B bus.
		
		`REGSB0 = IREGSB0`
		`REGSB1 = IREGSB1`
		`REGSB2 = IREGSB2`

REGSCn	3	This is the C-register select control bus. It is used by the Control Unit to tell the Register Unit which register to select onto the C bus.
		
		`REGSC0 = IREGSC0`
		`REGSC1 = IREGSC1`
		`REGSC2 = IREGSC2`

ALUn	4	This is the ALU control code bus. It is used by the Control Unit to tell the ALU which operation to perform.
		
		`ALU0 = IALU0`
		`ALU1 = IALU1`
		`ALU2 = IALU2`
		`ALU3 = IALU3`