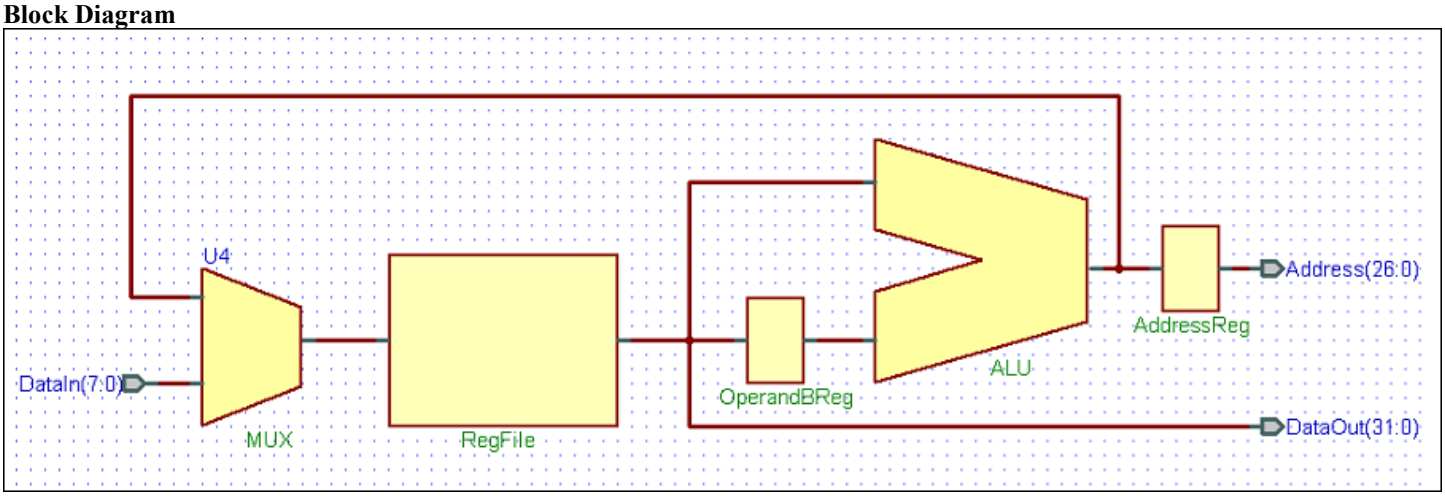
JRISC Draft Feb7 2004

Motivation

The JRISC processor is an attempt to make the *smallest* and most flexible 32-bit RISC processor for the Altera ACEX devices used on the C1 reconfigurable computer. The Verilog code is very Altera specific due to the limitations in the ACEX parts. (In other words this wont synthesize well on Xilinx parts).

Block Diagram



Theory of Operation

The JRISC processor differs from most pipelined RISC processors, because even from the beginning the goal was not execution speed, but low FPGA logic cell usage. Altera ACEX parts don't have large RAM blocks for caches, so the design will be bound by external memory and any time multiplexed devices. In the C1's case the memory is shared heavily with high-resolution VGA frame buffer. The processor takes 7 clock cycles to execute one instruction, which is faster than the C1's available memory bandwidth.

State Flow

Cycle 1	Fetch opcode and store in r15, latch flags from r14.
Cycle 2	Latch B operand from register file and decode conditions.
Cycle 3	If conditions true then pass A operand to ALU, write ALU result to destination (none, address register or register file) and if in load operation wait for memory access, then store memory data into register file.
Cycle 4	Latch r0 into B operand. If in store operation wait for memory access, then write data from register file.
Cycle 5	Write flags and current address to r14 if store flags field set in opcode.
Cycle 6	Latch B operand with sign bit [7] extended value from r15.
Cycle 7	Pass program counter (r13) to operand A and write result back to r13. Carry is set during address calculation so opcode [7:0] = 0 will increment the program counter. Offset is signed and relative.

Register File

Register	Description
0	This register must remain a constant value of 0. Use for several addressing modes and operations.
1	User
2	User
3	User
4	User
5	User
6	User
7	User
8	User
9	User
10	User
11	Bits [31:27] Reserved [26:0] IRQ Vector
12	Bits [31:27] Reserved [26:0] Software Stack
13	Bits [31:27] Reserved [26:0] Program Counter
14	Bit [31] Negative [30] Zero [29] Carry [28] Overflow [27] IRQ Enable [26:0] Last Address
15	Current Opcode Fetched. Modifying the [7:0] will change the offset that is calculated at end of the cycle (bad idea).

Negative	Bit 31 of the ALU result.
Zero	Set to 1 when ALU result is equal to 0.
Carry	Carry out from the ALU or left shifter.
Overflow	Detects addition of two signed binary numbers that overflow (i.e. two negative numbers may overflow)
IRQ Enable	Enables IRQ's when flag is set to 1.
Last address	Address of the last address that was store with the store flags opcode field. This value is not valid after a store operation.

Opcode Fields

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
Load	Conditions	1	Store Flags	01	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
Store	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Source Register	Operand B	Relative Branch Offset

Conditions

0000	EQ	(Equal)	Z
0001	NE	(Not Equal)	~Z
0010	CS	(Carry Set)	C
0011	CC	(Carry Clear)	~C
0100	MI	(MInus)	N
0101	PL	(PLus)	~N
0110	VS	(oVerflow Set)	V
0111	VC	(oVerflow Clear)	~V
1000	HI	(HIgher)	C and ~Z
1001	LS	(Lower or Same)	~C and Z
1010	GE	(Greater or equal)	N = V
1011	LT	(Less Than)	N = ~V
1100	GT	(Greater Than)	(N = V) and ~Z
1101	LE	(Less or equal)	(N = ~V) or Z
1110	AL	(Always)	True
1111	NV	(Never)	False

Store Result

0	Don't store ALU result
1	Store ALU result in register file

Store Flags

0	Don't update flags at end of execution or store current address.
1	Update flags at end of execution.

Opcodes

00	ALU Operation
01	Load Register
10	Store Registers
11	Reserved

ALU Functions

0000	AND
0001	OR
0010	XOR
0011	ADD + Carry
0100	AND Not B
0101	OR Not B
0110	XNOR

0111 SUB - Carry
1000 AND Shift Right
1001 OR Shift Right
1010 XOR Shift Right
1011 ADD + Carry Shift Right
1100 AND Extended B[7]*
1101 OR Extended B[7]*
1110 XOR Extended B[7]*
1111 ADD + Carry Extended B[7]*

Offset Branch
The relative location to jump after opcode is executed (only if condition is met)
00000000 = PC + 1
00000001 = PC + 2
11111111 = PC
11111110 = PC - 1

Code Examples:

And

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	True		AND	r1	r2	r3	000000000

Register 1 is “and-ed” with register 3, ALU result stored in register 2; flags/current address stored in r14 and program counter is incremented

ADD

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	True		ADD	r1	r2	r3	000000000

Register 1 is “added” with register 3, ALU result stored in register 2, flags/current address stored in r14 and program counter is incremented

Compare

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	False	True		SUB	r1	Don’t care	r3	000000000

Register 3 is “Subtracted” from register 1, flags/current address stored in r14 and program counter is incremented

Move

Bits	**31:28**	**27**	**26**	**25:24**	**23:20**	**19:16**	**15:12**	**11:8**	**7:0**
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	True		OR	r1	r2	r0	000000000

Moves register 1 to register 2. This operation can affect flags if needed.

Shift Left

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	True		ADD	r1	r2	r1	000000000

To save space in the ALU shift left was omitted an can be replaced with adding the same register with it's self. Here we shift r1 left one position, store the result and flags.

Conditional Branch

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	The Condition	False	False		Don't Care	Don't Care	Don't Care	Don't Care	000000011

If condition is true then branch forward 5 addresses else fetch next opcode.

Branch Always

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	False	False		Don't Care	Don't Care	Don't Care	Don't Care	11111110

Always branch backwards 1 address.

Signed Operations

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
ALU	Conditions	Store Result	Store Flags	00	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	True		AND	r1	r2	r3	000000000

Register 1 is "and-ed" with register 3 sign extended (Bit [7] fills all bit locations from [31:7] and [6:0] is passed unchanged), ALU result stored in register 2; flags/current address stored in r14 and program counter is incremented.

Load From Register Address

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
Load	Conditions	Store Result	Store Flags	01	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	False		OR	r1	r2	r0	000000000

Load memory value into r2 using r1 as the address pointer. Flags are not affected by load data, but from the address generation.

Load From ALU Result Address

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
Load	Conditions	Store Result	Store Flags	01	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	False		ADD	r1	r2	r3	000000000

Load memory value into r2 using (r1 + r3) as the address pointer. Flags are not affected by memory data, but from the address generation. Any ALU function can be used to generate address. Carry is not passed to the ALU during loads operations.

Load Immediate

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
Load	Conditions	Store Result	Store Flags	01	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	False		ADD Sign Extended	R13	r2	r15	000000001

Loads memory value from next address, then fetches opcode 2 addresses forward. Flags are not affected by memory data, but from the address generation. Carry is not passed to the ALU during Loads.

****Note About Store Commands****
You *CAN'T* return from a relative branch of a store opcode. The last address stored in r14 will be corrupted.

Store From Register Address

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
Store	Conditions	Store Result	Store Flags	10	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	False		OR	r1	r2	r0	000000000

Store memory value from r2 using r1 as the address pointer. Flags are not affected by stored data, but from the address generation.

Store From ALU Result Address

Bits	31:28	27	26	25:24	23:20	19:16	15:12	11:8	7:0
Store	Conditions	Store Result	Store Flags	10	ALU Operation	Operand A	Destination	Operand B	Relative Branch Offset
	Always	True	False		ADD	r1	r2	r3	000000000

Store memory value from r2 using (r1 + r3) as the address pointer. Flags are not affected by stored data, but from the address generation. Any ALU function can be used to generate address. Carry is not passed to the ALU during loads operations.

Startup
The first operation startup code should do for safety is load immediate $00000000 into r0, then set stack, IRQ vectors and last enable IRQ's

Jumping Absolute
Jumping is executed by loading or moving a value into r13 (program counter) with an offset of $FF. The offset keeps the program counter from being incremented.

Handling Subroutines

Notes:
Store Flags must be set in opcode that branches, so the last address register will be updated.
You *CAN'T* return from a store opcode. The last address register will be corrupt.

Entry:
The first opcode on branch entry must be store flags/last address (r14) to stack (r12 coding convention) or branch address will be lost.
Increment stack pointer and then start subroutine.

Exit:
Decrement stack pointer.
Load r14 with stack data.
Move r14 to r13 (program counter), then the processor will increment the program counter and return from subroutine.

Handling IRQ's

Notes:
IRQ's will only branch from opcodes that store flags/last address and not store operations.

Entry:

The first opcode on branch entry must be store flags/last address (r14) to stack (r12 coding convention) or branch address will be lost. Increment stack pointer and then start subroutine.

Exit:
Decrement stack pointer.
Load r14 with stack data. (This will enable IRQ's by setting the IRQ flag back to 1)
Move r14 to r13 (program counter), then the processor will increment the program counter and return from subroutine.

Useful Reading:

<Put some junk here>