Glacial Microarchitecture

Eric Smith

October 28, 2019

Chapter 1

Introduction

This document describes the microarchitecture of Glacial, a highly vertically-microcoded processor. When running the standard microcode, Glacial implements a RISC-V RV32I processor. However, the Glacial microarchitecture may be useful for other applications unrelated to RISC-V.

In the rest of this document, the use of the name "Glacial" will be in reference to the Glacial microarchitecture, without regard to the normal microcode implementing the RISC-V architecture.

1.1 Limitations

Glacial was designed to use minimal hardware resources, e.g., FPGA logic elements, and was designed to support hand-written assembly code, rather than a high-level language. It is missing many features commonly present in other processor architectures. For example:

- There is an adc instruction (add with carry), but no subtract instruction, because subtraction can be accomplished by complementing the subtrahend then adding.
- There are instructions for logical and and exclusive or, but not for logical or. By De Morgan's law, the logical and and exclusive or instructions can be used to accomplish a logical or function.
- There is a postincrement address mode, but no corresponding predecrement mode
- Glacial has a single memory address space of 64 KiB. Glacial can only execute instructions from the first 4 KiB of the memory; the remaining 60 KiB is used for data (or macroinstructions) only.
- The X index register, and instructions using absolute addressing of data, can only address the first 256 bytes of memory. The rest of memory can only be accessed as data by use of the Y index register.

- There is only one level of subroutine stack.

1.2 Simple architectural enhancements

Several relatively simple architectural enhancements could be made, which might make Glacial more suitable for other applications:

- More index registers could be added. The instruction encoding for the indexed addressing modes currently does not use bits 7..1 of the instruction.
- The address space for data could be increased, e.g., to 16 MiB or 4 GiB, by making the Y index register (and/or other added index registers) wider.
- The size of the return stack could be increased to allow multiple levels of subroutine calls.
- If it is desired to use ROM starting at address 0, the X index register could have a fixed but non-zero high byte to address RAM. This would be similar to the 6502 microprocessor's stack pointer, which is 8 bits, but has a high byte of 0x01.

Chapter 2

Programmer's Model

Glacial has six programmer-visible registers:

register	bits
program counter	12 (LSB always zero)
return address	12 (LSB always zero)
x index	8
y index	16
accumulator	8
carry flag	1

Glacial has ten basic instructions, though one of them, opr (operate) performs various miscellaneous functions:

mnemonic	description
opr	operate (miscellaneous functions)
store	store the accumulator into a memory byte
load	load a byte from memory into the accumulator
and	logically and a byte from memory into the accumulator
xor	logically exclusive or a byte from memory into the accumulator
adc	add a byte from memory and carry into the accumulator
jump	unconditional jump
call	unconditional subroutine call
skb	skip next instruction, conditional on the state of an accumulator bit
br	conditional branch

Glacial has five basic memory addressing modes, not all of which are applicable to all instructions:

- immediate 8-bit

- absolute 8-bit
- index register
- index register with postincrement
- branch, absolute 13-bit address, even addresses only

Chapter 3

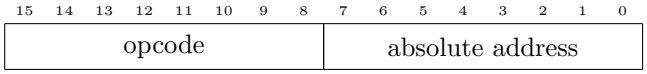
Instruction Formats and Memory Addressing Modes

3.1 immediate 8-bit

The immediate addressing mode embeds an 8-bit operand direction in the instruction.

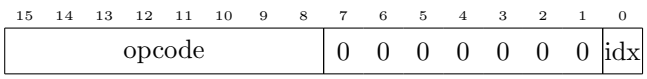
3.2 absolute 8-bit

The absolute addressing mode allows access to any byte within the first 256 bytes of memory (0x0000 through 0x00ff) by a fixed address embedded in the instruction.



3.3 index register

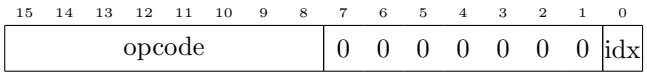
The indexed addressing mode allows access to the byte of memory pointed to by an index register, x or y. The x index register can only point to the first 256 bytes of memory (0x0000 through 0x00ff).



The idx field value is 0 for the x index register, and 1 for the y index register.

3.4 index register autoincrement

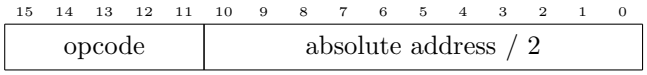
The index register autoincrement addressing mode allows access to the byte of memory pointed to by an index register, x or y, and after the access, increments the index register. The x index register can only point to the first 256 bytes of memory (0x0000 through 0x00ff), and if incremented past 0xff, x will wrap to 0x0000.



The idx field value is 0 for the x index register, and 1 for the y index register.

3.5 branch

The branch instructions include 11 bits of a 12-bit absolute address of the branch target. Since instructions must be 16-bit aligned, the branch target absolute address is always even, so the LSB is omitted from the instruction encoding.

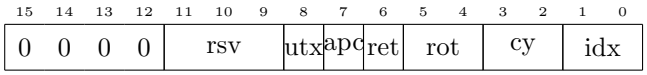


Chapter 4

Instructions

4.1 opr: operate

The opr (operate) instruction is a catch-all for miscellaneous functions, with a bit-mapped opcode containing bits or bitfields encoding those functions. This allows the combination of multiple functions into a single instruction. This technique was used in the 1960s and 1970s by the Digital Equipment Corporation 18-bit and 12-bit minicomputers such as the PDP-1 and PDP-8, where it was called "microcoding" (similar to horizontal microcode).



bits	mnemonic	function
11..9	reserved	
8	utx	UART transmit LSB of accumulator
7	apc	add accumulator to PC
6	ret	return from subroutine
5..4	00 = nop, 10 = rlc, 11 = rrc	rotate left or right with carry
3..2	00 = nop, 10 = clc, 11 = sec	clear or set carry
1..0	00 = nop, 10 = tax, 11 = tay	transfer accumulator to index register

Some useful combinations:

opcode	functions	mnemonic	function
0x028	clc rol	lsl	logical shift left accumulator
0x038	clc ror	lsr	logical shift right accumulator
0x0c0	ret apc	retadd	return skipping word count in accumulator
0x13c	utx sec rrc	uarttxr	UART transmit LSB of accumulator

The carry operations are executed before the rotate operations.

The tax and tay operations transfer the accumulator contents to the index register. Since x is an 8-bit register, tax simply performs a copy. The y register, on the other hand, is a 16-bit register, so in order to allow loading the entire y register, each use of the tay operation shifts y right by eight bits and puts the accumulator value into the high 8 bits of y.

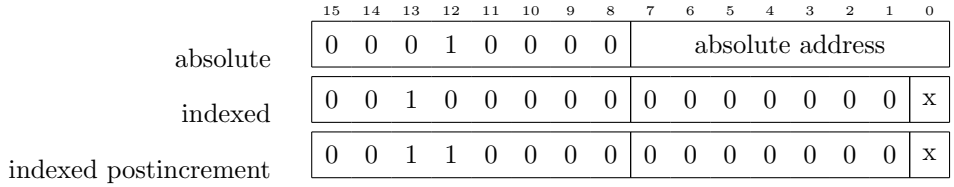
For example, if the y register contains 0x1234 and the accumulator contains 0x78, after one tay instruction the y register will contain 0x7812.

The utx operation is specifically intended for use in the uarttxr combination, which is used for a bit-banged UART transmit output. This was particularly useful for the RISC-V conformance tests, but may not be suitable for a real application. The uarttxr instruction should be used once with the LSB of the accumulator containing zero, to generate a start bit, then the byte to be transmitted should be loaded into the accumulator, and uarttxr executed nine more times, with appropriate delays between uarttxr instructions for the serial bit timing desired.

The apc operation is useful for jump tables. Combined with ret to form the retadd instruction, it is useful as a return from subroutines that have multiple exits, with the accumulator specifying a number of instructions to skip beyond the normal return location.

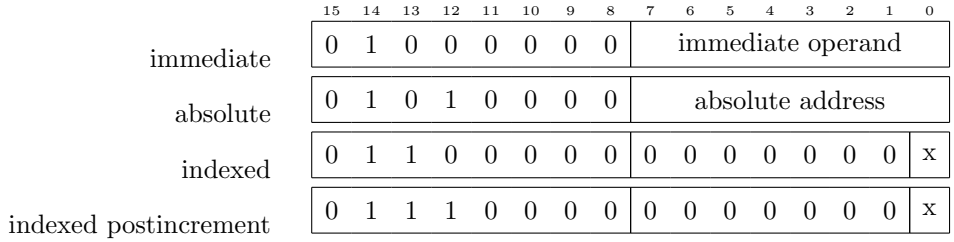
4.2 store: store accumulator to memory

The store instruction is used to store the contents of the accumulator register into a byte of memory.



4.3 load: load accumulator from memory

The load instruction is used to load the contents of a byte of memory into the accumulator register.



4.4 and: and memory with accumulator

The and instruction is used to logically and the contents of a byte of memory into the accumulator register.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
immediate	0	1	0	0	0	0	0	1	immediate operand							
absolute	0	1	0	1	0	0	0	1	absolute address							
indexed	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	x
indexed postincrement	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	x

4.5 xor: exclusive or memory with accumulator

The xor instruction is used to logically exclusive-or the contents of a byte of memory into the accumulator register.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
immediate	0	1	0	0	0	0	1	0	immediate operand							
absolute	0	1	0	1	0	0	1	0	absolute address							
indexed	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	x
indexed postincrement	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	x

4.6 adc: add memory to accumulator with carry

The adc instruction is used to add the contents of a byte of memory and the carry flag into the accumulator register, and will set the carry flag based on the result of the addition.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
immediate	0	1	0	0	0	0	1	1	immediate operand							
absolute	0	1	0	1	0	0	1	1	absolute address							
indexed	0	1	1	0	0	0	1	1	0	0	0	0	0	0	0	x
indexed postincrement	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	x

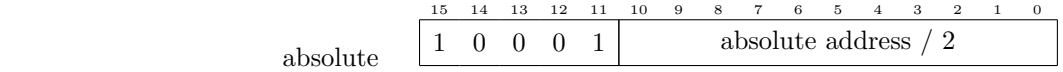
4.7 jump: unconditional jump

The jump instruction performs an unconditional jump to the target address.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
absolute	1	0	0	0	0	absolute address / 2										

4.8 call: unconditional subroutine call

The call instruction pushes the address of the next sequential instruction into the return register, and branches to the target address.

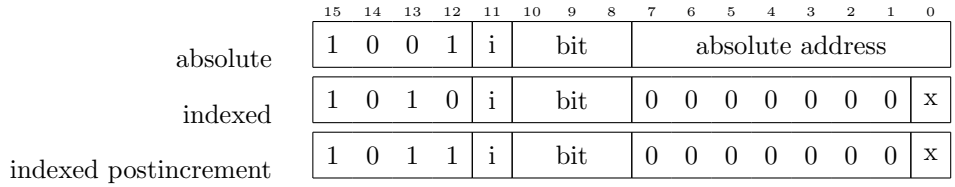


	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
absolute	1	0	0	0	1	absolute address / 2										

Glacial normally has only one level of return stack, so nested subroutine calls will not work.

Returning from a subroutine is accomplished by the "ret" function of the opr (operate) instruction.

4.9 skb: skip on bit

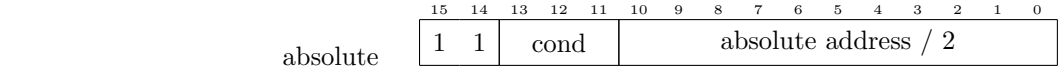


	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
absolute	1	0	0	1	i	bit			absolute address							
indexed	1	0	1	0	i	bit			0	0	0	0	0	0	0	x
indexed postincrement	1	0	1	1	i	bit			0	0	0	0	0	0	0	x

The "i" bit determines the polarity of the bit test and the "bit" field determines which bit of the operand is tested, with 0 being the least significant bit. The skip will occur if the "i" bit matches the bit being tested.

4.10 br: conditional branch

The conditional branch instruction will branch to the target address if the condition is true. Otherwise execution will continue sequentially.



	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
absolute	1	1	cond			absolute address / 2										

The "cond" field determines which condition is tested.

cond	condition
0	accumulator non-zero
1	accumulator zero
2	no carry
3	carry
4	no external interrupt
5	external interrupt
6	no clock tick
7	clock tick