What is Sherwood Architecture?

The Sherwood Architecture is a custom 64-Bit RISC based CPU architecture. It is designed to be lightweight, fast, and powerful. This repository comes with the circuit schematics, compilers, debug and boot firmware, and a virtual machine for running the CPU. Our goal is also aimed to people interested in programming and hardware design so we've designed the architecture to be friendly to everyone. This document outlines the hardware and design of the architecture.

CPU

The CPU of the Sherwood Architecture is 64-Bits which is nice for lots of computing and work. The latest revision of the CPU has 9 interrupts, 49 registers, 52 instructions, and can address up to 6917.5 Petabytes of RAM since the start vector is the beginning of physical RAM and is mapped to 0xA0000000. The address and data busses are both 64 bits so there's no need to worry about not having enough bits. Enabling IRQ's requires setting the IRQ flag (bit 1) to enabled.

Flags

This is a table of all of the flags in the CPU's flags register.

Bits	Name	Description
0	INTR	Is the CPU in an interrupt? Cannot be edited.
1	ENIRQ	Enables IRQ's.
2	PAGING	Enables paging. Cannot be edited in userspace.
3	PRIV_KERN	Enables kernel privilages. Cannot be edited in userspace.
4	PRIV_USER	Enables userspace privilages. Cannot be edited in userspace.

Interrupts:

- **STACK_OVERFLOW** (0): This interrupt occurs when there's a stack overflow from the push, pop, and call instructions.
- **FAULT** (1): Whenever this instruction occurs, a fault has occurred.
- **BADADDR** (2): Whenever the program tries to access registers or memory locations that don't exist then this interrupt occurs.
- **DIVBYZERO** (3): Caused by trying to divide by zero.
- **BADINSTR** (4): This interrupt occurs when an invalid instruction gets called.
- **BADPERM** (5): This interrupt occurs when the program doesn't have the correct permissions for something.
- **TIMER** (6): The RTC triggers this interrupt when a timer has finished.

- **MAILBOX** (7): The interrupt that the mailbox uses. Look at the mailbox's page for more information.
- **SYSCALL** (8): This interrupt that programs use to communicate with the operating system.

Registers:

- **flags** (0): Stores special flags like whether the CPU is in an interrupt or not.
- **tmp** (1): Temorary data
- **sp** (2): The current stack pointer
- **ip** (3): The current instruction pointer
- **pc** (4): The program counter
- **cycle** (5); The current cycle count
- **data#** (6-16): The data register, replace # with a number 0-9 to access one of the data registers**.**
- **index#** (17-27): The index register, replace # with a number 0-9 to access one of the index registers.
- **addr#** (28-38): The address register, replace # with a number 0-9 to access one of the index registers.
- **ptr#** (39-49): The pointer register, replace # with a number 0-9 to access one of the pointer registers.

Instructions:

Name	Description	Usage
nop	No operation (alias of hlt).	nop
addr	Add numbers from register to register.	addr %register,%register
addm	Add numbers from memory to memory.	addm $address,$address
subr	Subtracts numbers from register to register.	subr %register,%register
subm	Subtracts numbers from memory to memory.	subm $address,$address
mulr	Multiplies numbers from register to reigister.	mulr %register,%register

mulm	Multiplies numbers from memory to memory.	mulm $address,$address
divr	Divides numbers from register to register.	divr %register,%register
divm	Divides numbers from memory to memory.	divm $address,$address
andr	Bitwise AND operation from register to register.	andr %register,%register
andm	Bitwise AND operation from memory to memory.	andm $address,$address
orr	Bitwise OR operation from register to register.	orr %register,%register
orm	Bitwise OR operation from memory to memory.	orm $address,$address
xorr	Bitwise XOR operation from register to register,	xorr %register,%register
xorm	Bitwise XOR operation from memory to memory.	xorm $address,$address
norr	Bitwise NOR operation from register to register.	norr %register,%register
norm	Bitwise NOR operation from memory to memory.	norm $address,$address
nandr	Bitwise NAND operation from register to register.	nandr %register,%register
nandm	Bitwise NAND operation from memory to memory.	nandm $address,$address
lshiftr	Bitwise left shift operation from register to register.	lshiftr %register,%register
lshiftm	Bitwise left shift operation from memory to memory.	lshiftm $address,$address
rshiftr	Bitwise right shift operation from register to register.	rshiftr %register,%register
rshiftm	Bitwise right shift operation from memory to memory.	rshiftm $addrress,$address
cmpr	Compare registers, sets the tmp register to true if the values are equal.	cmpr %register,%register
cmpm	Compare memory addresses, sets the tmp register to true if the values are equal.	cmpm $address,$address
jitr	Jumps to a location from a register if the tmp register is equal to 1 (true).	jitr %register
jitm	Jumps to a location from a memory address if the tmp register is equal to 1 (true).	jitm $address

jit	Jumps to a location if the tmp register is equal to 1 (true).	jit #value
jmpr	Jumps to a memory location from register value.	jmpr %register
jmpm	Jumps to a memory location from memory.	jmpm $address
jmp	Jumps to a memory location.	jmp #value
callr	Calls a memory location from a register.	callr %register
callm	Calls a memory location from a memory address.	callm $address
call	Calls a memory location.	call #value
ret	Returns from the called memory location.	ret
pushr	Pushes the value from a register to the stack.	pushr %register
pushm	Pushes the value from a memory location to the stack.	pushm $address
popr	Pops the top value from the stack and stores it in a register.	popr %register
popm	Pops the top value from the stack and stores it in memory.	popm $address
movrr	Moves the value from a register to another register.	movrr %register,%register
movrm	Moves the value from a register to memory.	movrm %register,$address
movmr	Moves the value from memory to a register.	movmr $address,%register
movmm	Moves the value from memory to another location in memory.	movmm $address,$address
stor	Stores a value in a register.	stor %register,#value
stom	Stores a value in memory.	stom $address,#value
intr	Runs an interrupt, the interrupt number is read from a register. Only works with kernel privilages.	intr %register
intm	Runs an interrupt, the interrupt number is read from memory. Only works with kernel privilages.	intm $address
int	Runs an interrupt. Only works with kernel privilages.	int #value

iret	Returns from an interrupt. Only works with kernel privilages.	iret
lditblr	Loads the IVT from a memory address read from a register. Only works with kernel privilages.	lditblr %register
lditblm	Loads the IVT from a memory address. Only works with kernel privilages.	lditblm $address
hlt	Alias of nop. Only works with kernel privilages.	hlt
rst	Resets the CPU. Only works with kernel privilages.	rst

Paging

The paging system is quite simple on the Sherwood Architecture. Setting or clearing the second bit in the flags register will enable or disable paging but it only works after you load the page directory into the IO Controller.

Page

This is the layout of a page.

Size (Bytes)	Offset (Bytes)	Name	Description
4	0	flags	The flags that's set in the page.
4	4	perms	The page's permissions.
8	8	size	The size of the page.
8	16	address	The address that the page is mapped to.

Page Table

The layout of a page table.

Size (Bytes)	Offset (Bytes)	Name	Description
8	0	size	The number of pages in the table.
24*size	8	entries	An array/list of pages.

Page Directory

The layout of a page directory, this is loaded into the IO Controller before enabling paging on the CPU.

Size (Bytes)	Offset (Bytes)	Name	Description
8	0	size	The number of page tables in the directory.
(8+(24*size of pages))*size of table	8	tables	An array/list of tables.

Permissions

This is a table of bits in the permissions variable in the page structure.

Bits	Name	Description
0	KERN	Enable kernel level access.
1	USER	Enable userspace level access.

Flags

This is a table of bits in the flags variable in the page structure.

Bits	Name	Description
0	PRESENT	Is the page present?
1	ACCESSED	Has the page been accessed?
2	DIRTY	Is the page dirty?

IO Controller

The IO Controller for the Sherwood Architecture manages the memory map and access to memory. This is one of the most simplest components of the Sherwood Architecture.

Memory Map:

Address	Size	Description
0x10000000	0x00000009	Mailbox Memory
0x1000000A	0x00000007	RTC (Real Time Clock) Memory
0x10000012	0x00000003	UART Memory
0x10000016	0x00000005	IO Controller Memory
0xA0000000	[Physical RAM Size]	Physical RAM installed

Address Space:

Address	Description	Access
0	Get the ammount of physical RAM installed.	RO
1	Sets the Page Directory	WO
2	Read: Get CPU core count. Write: select the core to interact with.	RW
3	Read: Get the current CPU core. Write: sets the running state.	RW
4	Read: Get selected core to interact with. Write: sets the program counter on the selected core.	RW

Mailbox

The mailbox is the device bus on the Sherwood Architecture and it does not support plug and play, it's similar to the Raspberry Pi's mailbox as it is a bus for devices that is accessed from memory. When the mailbox sends an interrupt, the data0 register is set to the device's index.

Address Space:

These are the addresses within the mailbox's memory mapped location.

Address	Description	Access
0x0	Returns the device count.	RO
0x1	Get/Set the current device to access.	RW
0x2	Get/Set the device's data index.	RW
0x3	Read/Write to the device's data.	RW
0x4	Get the device's vendor ID. This is provided by us, please contact us in order to get a valid vendor ID.	RO
0x5	Get the device's device ID. This is whatever the manufacturer decided the device's ID to be set to.	RO
0x6	Get the device's revision.	RO
0x7	Get the device's type. The types are standardized by us and a list can be seen in the next table.	RO
0x8	Get the device's class code. The class codes are standardized by us and a list can be seen in the table after the device type table.	RO
0x9	Get the device's subclass. This is the device's version code.	RO

Device Types

This table shows every device type for the mailbox.

Code	Name	Description
0	NET	Networking Devices
1	STORAGE	Storage Devices
2	MULTIMEDIA	Multimedia Devices (Camera, capture cards, etc)

3	BUSES	Bus Devices (USB, PCI, AGP, etc.)
4	GRAPHICS	Graphical Devices
5	SOUND	Sound Devices
6	MISC	Miscellaneous Devices

Device Class Codes

Each type of device has a subcategory, the devices class code.

Network Devices Class Codes

Class	Name	Description
0	ETH	Ethernet Card
1	WIFI	WiFi Card
2	BT	Blutooth Card

Storage Devices Class Codes

Class	Name	Description
0	HDD	Hard Disk Drive
1	DVD	DVD Drive
2	BR	BluRay Drive

Multimedia Devices Class Codes

Class	Name	Description
0	CAP	Capture Card
1	CAM	Camera

Buss Devices Class Codes

Class	Name	Description
0	USB	USB Bus
1	PCI	PCI Bus

Graphical Devices Class Codes

Class	Name	Description
0	GC	Graphics Card
1	3DGC	3-D Graphics Accelerator

Sound Devices Class Codes

Class	Name	Description
0	SND	Sound Card
1	MIDI	Midi Card

Miscellaneous Devices Class Codes

Class	Name	Description
0	CPUACCEL	CPU Accelerator
1	MEM	Memory Expansion
2	ROM	A ROM device
3	MEMCTL	A Memory Controller (virtual memory controller, another IO Controller, etc.)
4	CON	A console like device. (UART, COM, LPT, etc.)

RTC

The system's RTC chip (Real Time Clock), this handles all timers and time related stuff.

Address Space:

Address	Description	Access
0	Seconds	RW
1	Minutes	RW
2	Hours	RW
3	Day of month	RW
4	Month	RW
5	Year	RW
6	Is DST Observed right now?	RO
7	Set: Creates a new timer. Get: Returns timer count.	RW

UART

The system's UART chip, this is the simplest chip ever on the Sherwood Architecture.

Address Space:

Address	Description	Access
0	Is there input?	RO
1	Get: inputted character. Set: outputs character.	RW
2	Is output ready?	RO