# Thor II

Dropped from the Thor design:

* variable length instructions.
* instruction predicates / predicate registers
* code address registers

Variable length instructions added some complexity to the instruction fetch stage of the processor including also the instruction cache. Variable length instructions helped improve Thor’s code density.

Instruction predicates proved to be not that useful given the size of the instruction queue. The compiler was only able to emit predicated instructions on rare occasions. It is desirable to support the common case with hardware, not the rare case. The instruction queue would have to be much larger for instruction predicates to become more useful.

Improved from Thor

The instruction cache is two level now (L1, L2) allowing better performance. The first level cache is fully associative, the second level cache is four-way set associative.

# Instruction Set Description

Instructions have a fixed 32 bit format. Immediate constants may be extended using prefix instructions.

# ADD

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 04h6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 046 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

Clock Cycles: 0.5

# AND – Bitwise And

Description:

Perform a bitwise and operation between two operands. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 08h6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 086 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

Clock Cycles: 0.5

# ASR – Arithmetic Shift Right

Description:

Bits from the source register Ra are shifted right by the amount in register Rb or an immediate value. The sign bit is shifted into the most significant bits.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 146 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 156 | ~4 | I1 | Rt5 | Imm5 | Ra5 | 02h6 |

Clock Cycles: 1

# BEQ/BNE/BMI/BPL – Conditional Branch

Description:

If the branch condition is true, a sixteen bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch. The immediate value may not be extended with a prefix instruction.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed16 | P2 | Cond3 | Ra5 | 04h6 |

|  |  |  |
| --- | --- | --- |
| Cond3 | Mne. |  |
| 0 | BEQ | register Ra = 0 |
| 1 | BNE | register Ra <> 0 |
| 2 | BMI | register Ra < 0 (bit 63 is set) |
| 3 | BPL | register Ra >=0 (bit 63 is clear) |
| 4-7 |  | reserved |

The P2 field is reserved for branch prediction hints.

|  |  |
| --- | --- |
| P2 | Prediction Type |
| 0 | no static prediction (use branch history) |
| 1 | reserved |
| 2 | always predict as not-taken |
| 3 | always predict as taken |

If a branch prediction is supplied, then the branch instruction doesn’t occupy room in the history tables.

# BRK – Hardware / Software Breakpoint

|  |  |  |  |
| --- | --- | --- | --- |
| 31 16 | 15 | 14 6 | 5 0 |
| Immed16 | H | Vector9 | 00h6 |

H = 0 = software interrupt – return address is next instruction

H = 1 = hardware interrupt – return address is current instruction

# CLI – Clear Interrupt Mask

CLI

Description:

The interrupt level mask is set to zero enabling all interrupts.

Instruction Format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 316 | ~2 | ~3 | ~5 | ~5 | ~5 | 02h6 |

Clock Cycles: 0.5

# CMP – Signed Comparison

Description:

The compare instruction places a 1, 0 or -1 in the target register based on the relationship between the two source operands. If they are equal a zero is placed in the target register, if register Ra is less than the second operand then a -1 is placed in the target register, otherwise a 1 is placed in the target register. The values are treated as signed operands.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 06h6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 066 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

Clock Cycles: 0.5

# CMPU – Unsigned Comparison

Description:

The compare instruction places a 1, 0 or -1 in the target register based on the relationship between the two source operands. If they are equal a zero is placed in the target register, if register Ra is less than the second operand then a -1 is placed in the target register, otherwise a 1 is placed in the target register. The values are treated as unsigned operands.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 07h6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 076 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

Clock Cycles: 0.5

# CSR – Control and Status Access

Description:

The CSR instruction group provides access to control and status registers in the core. For the read-write operation the current value of the CSR is placed in the target register Rt then the CSR is updated from register Ra.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op2 | OL2 | Regno12 | Rt5 | Ra5 | 0Eh6 |

|  |  |  |
| --- | --- | --- |
| Op2 |  | Operation |
| 0 | CSRRD | Only read the CSR, no update takes place, Rt should be 0. |
| 1 | CSRRW | Both read and write the CSR |
| 2 | CSRRS | Read CSR then set CSR bits |
| 3 | CSRRC | Read CSR then clear CSR bits |

CSRRS and CSRRC operations are only valid on registers that support the capability.

The OL2 field is reserved to specify the operating level.

|  |  |  |  |
| --- | --- | --- | --- |
| Regno12 |  | Access | Description |
| 001 | HARTID | R | hardware thread identifier (core number) |
| 002 | TICK | R | tick count, counts every cycle from reset |
| 040 | EPC | RW | exceptioned pc, pc value at point of exception |
| 044 | STATUS | RWSC | status register, contains interrupt mask, operating level |

Clock Cycles: 0.5

# IMM – Immediate Prefix

Description:

The immediate prefix instruction extends the immediate constant of the following instruction. Immediate constants up to 64 bits may be formed by using two immediate constant prefixes in succession. If using two prefixes the low order prefix should appear first in the instruction stream. The assembler will automatically emit prefix instructions where needed.

Instruction Format:

|  |  |
| --- | --- |
| Immediate[41..16] | 1Ah6 |

|  |  |  |
| --- | --- | --- |
| ~4 | Immediate[63..42] | 1Bh6 |

Clock Cycles: 0.5

# JAL – Jump-And-Link

Description:

This instruction loads the program counter with the sum of a register and a constant value specified in the instruction. In addition the address of the instruction following the JAL is stored in the specified target register. This instruction may be used to implement subroutine calls and returns.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 18h6 |

Clock Cycles:

# LH – Load Half-Word

Description:

This instruction loads a half-word (32 bit) value from memory. The memory address must be half-word aligned. The value is sign extended to 64 bits when placed in the target register.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 10h6 |

Clock Cycles: 4 minimum depending on memory access time

# LHU – Load Half-Word

Description:

This instruction loads a half-word (32 bit) value from memory. The memory address must be half-word aligned. The value is zero extended to 64 bits when placed in the target register.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 11h6 |

Clock Cycles: 4 minimum depending on memory access time

# LW – Load Word

Description:

This instruction loads a word (64 bit) value from memory. The memory address must be word aligned.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 12h6 |

Clock Cycles: 4 minimum depending on memory access time

# NOP – No Operation

Description:

The NOP instruction doesn’t perform any operation. NOP’s are detected in the instruction fetch stage of the core and are not enqueued by the core. They do not occupy queue slots. Because NOPs don’t occupy queue slots they may not be used to synchronize operations between instructions.

Instruction Format:

|  |  |
| --- | --- |
| Immediate26 | 1Ch6 |

# OR – Bitwise Or

Description:

Perform a bitwise or operation between two operands. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 09h6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 096 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

Clock Cycles: 0.5

# SEI – Set Interrupt Mask

SEI #3

Description:

The interrupt level mask is set to the value specified by the instruction. The assembler assumes a mask value of seven, masking all interrupts, if no mask value is specified.

Instruction Format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 306 | ~2 | M3 | ~5 | ~5 | ~5 | 02h6 |

# SH – Store Half-Word

Description:

This instruction stores a half-word (32 bit) value to memory. The memory address must be half-word aligned.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 14h6 |

Clock Cycles: 4 minimum depending on memory access time

# SHL – Shift Left

Description:

Bits from the source register Ra are shifted left by the amount in register Rb or an immediate value. Zeros are shifted into the least significant bits.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 106 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 126 | ~4 | I1 | Rt5 | Imm5 | Ra5 | 02h6 |

Clock Cycles: 1

# SHR – Shift Right

Description:

Bits from the source register Ra are shifted right by the amount in register Rb or an immediate value. Zeros are shifted into the most significant bits.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 116 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 136 | ~4 | I1 | Rt5 | Imm5 | Ra5 | 02h6 |

Clock Cycles: 1

# SW – Store Word

Description:

This instruction stores a word (64 bit) value to memory. The memory address must be word aligned.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 16h6 |

Clock Cycles: 4 minimum depending on memory access time

# SYNC -Synchronize

Description:

All instructions before the SYNC are completed and committed to the architectural state before instructions after the SYNC are issued. This instruction is used to ensure that the machine state is valid before subsequent instructions are executed.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 366 | ~5 | ~5 | ~5 | ~5 | 02h6 |

Clock Cycles: varies depending on queue contents

# XOR – Bitwise Exclusive Or

Description:

Perform a bitwise exclusive or operation between two operands. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| Immed16 | Rt5 | Ra5 | 0Ah6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0A6 | ~5 | Rt5 | Rb5 | Ra5 | 02h6 |

Clock Cycles: 0.5

# Opcode Tables

## Major Opcode (inst. bits 0 to 5)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK | Bxx | {RR} |  | ADDI |  | CMPI | CMPUI | ANDI | ORI | XORI |  |  |  | CSR |  |
| 1x | LH | LHU | LW |  | SH |  | SW |  | JAL |  | IMM | IMM | NOP |  |  |  |
| 2x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## Major Funct (inst. bits 26 to 31)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  |  |  |  | ADD | SUB | CMP | CMPU | AND | OR | XOR |  |  |  |  |  |
| 1x | SHL | SHR | SHLI | SHRI | ASR | ASRI |  |  |  |  |  |  |  |  |  |  |
| 2x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3x | SEI | CLI | RTI |  | MEMDB | MEMSB | SYNC |  |  |  |  |  |  |  |  |  |

# Appendix

Architectural Register vs Physical Registers

Architectural registers are the registers visible to the programmer as part of the programming model. Physical registers are the registers physically present in the machine’s hardware. There are substantially more physical registers than there are architectural ones. There are 32 registers visible to be programmed which are supported by 64 physical registers.

Register Renaming

The core maintains an eight entry deep history file for register rename mappings and register in use flags. The depth of the history file corresponds to the number of entries in the re-order buffer. At most a new map will be needed for each re-order buffer entry. Typically the history file is cycled through at half or less the rate of the instruction queue as approximately 50% of instructions don’t have target registers.

The core can allocate up to two registers as target registers for every pair of instructions queued. If there are no target registers available the core stalls until previous instructions have made more target registers available.

Instruction Cache Miss

During a cache miss the core streams NOP operations to the instruction fetch unit while the core is waiting for the instruction cache to load. The program counters are not incremented however, and they remain at the value when the cache miss occurred.

## Instructions Supported Only on ALU #0

The following less frequently used instructions are only supported on ALU #0 in order to reduce the size of the core.

* + shift instructions (ASR, SHL, SHR)
    - The shift instructions use barrel shifters to shift by any amount in a single clock cycle.
  + CSR instruction
    - CSR instructions are rarely used. They often also have synchronization issues as there is no bypassing for the CSR registers. Since they typically require synchronization operations there is no benefit to having multiple CSR instructions executing at the same time.