**Ace 21064 Microprocessor**

Architecture Specification

**r1p0**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Description** | **Author** |
| 1.0 | 10/13/2016 | Initial version | Yijun LI |
|  |  |  |  |

**Content**

[1 Ace 21064 Core Basics 7](#_Toc481705110)

[1.1 Overview 7](#_Toc481705111)

[1.2 Features 7](#_Toc481705112)

[1.3 Components 8](#_Toc481705113)

[1.4 Pipeline architecture of Ace21064 8](#_Toc481705114)

[2 Level 1 Instruction Cache 9](#_Toc481705115)

[2.1 Overview 9](#_Toc481705116)

[2.1.1 Introduction 9](#_Toc481705117)

[2.1.2 Architecture 10](#_Toc481705118)

[2.1.3 Signals 10](#_Toc481705119)

[2.1.4 Submodules 11](#_Toc481705120)

[2.1.5 Timing 11](#_Toc481705121)

[2.2 Instruction Alignment Unit 11](#_Toc481705122)

[2.2.1 Introduction 11](#_Toc481705123)

[2.2.2 Architecture 11](#_Toc481705124)

[2.2.3 Signals 11](#_Toc481705125)

[2.2.4 Submodules 11](#_Toc481705126)

[2.2.5 Timing 11](#_Toc481705127)

[3 Instruction Fetch Unit (IFU) 12](#_Toc481705128)

[3.1 Overview 12](#_Toc481705129)

[3.1.1 Introduction 12](#_Toc481705130)

[3.1.2 Signals 12](#_Toc481705131)

[3.1.3 Submodules 13](#_Toc481705132)

[3.2 Branch Prediction Unit 15](#_Toc481705133)

[3.2.1 Introduction 15](#_Toc481705134)

[3.2.2 Architecture 15](#_Toc481705135)

[3.2.3 Ports 15](#_Toc481705136)

[3.2.4 Timing 16](#_Toc481705137)

[3.2.5 Branch History Table (BHT) 16](#_Toc481705138)

[3.2.6 Pattern History Table (PHT) 18](#_Toc481705139)

[3.3 Return Address Stack (RAS) 19](#_Toc481705140)

[3.4 Branch Target Buffer (BTB) 19](#_Toc481705141)

[3.4.1 Introduction 19](#_Toc481705142)

[3.4.2 BTB Way 19](#_Toc481705143)

[3.4.3 Ports 19](#_Toc481705144)

[3.5 PC Generator 20](#_Toc481705145)

[4 Instruction Decoder Unit(IDU) 21](#_Toc481705146)

[4.1 Overview 21](#_Toc481705147)

[4.1.1 Introduction 21](#_Toc481705148)

[4.1.2 Signals 21](#_Toc481705149)

[4.1.3 Submodules 22](#_Toc481705150)

[4.2 Instruction Buffer 22](#_Toc481705151)

[4.2.1 Introduction 22](#_Toc481705152)

[4.2.2 Architecture 22](#_Toc481705153)

[4.2.3 Ports 22](#_Toc481705154)

[4.3 Instruction Decoder 23](#_Toc481705155)

[4.3.1 Introduction 23](#_Toc481705156)

[4.3.2 Architecture 23](#_Toc481705157)

[4.3.3 Ports 23](#_Toc481705158)

[5 Instruction Renaming Unit 26](#_Toc481705159)

[5.1 Overview 26](#_Toc481705160)

[5.1.1 Introduction 26](#_Toc481705161)

[5.1.2 Signals 26](#_Toc481705162)

[5.1.3 Submodules 27](#_Toc481705163)

[5.2 Instructions Renaming Unit 27](#_Toc481705164)

[5.2.1 Introduction 27](#_Toc481705165)

[5.2.2 Architecture 27](#_Toc481705166)

[5.2.3 Ports 28](#_Toc481705167)

[5.3 Memory dependency predictor 28](#_Toc481705168)

[5.3.1 Introduction 28](#_Toc481705169)

[5.3.2 Architecture 28](#_Toc481705170)

[5.3.3 Ports 28](#_Toc481705171)

[6 Instruction Schedule Unit(ISU) 29](#_Toc481705172)

[6.1 Overview 29](#_Toc481705173)

[6.1.1 Introduction 29](#_Toc481705174)

[6.1.2 Signals 29](#_Toc481705175)

[6.1.3 Submodules 30](#_Toc481705176)

[7 Execution Unit 32](#_Toc481705177)

[7.1 Overview 32](#_Toc481705178)

[7.1.1 Introduction 32](#_Toc481705179)

[7.1.2 Signals 32](#_Toc481705180)

[7.1.3 Submodules 33](#_Toc481705181)

[8 Level 1 data Cache 34](#_Toc481705182)

[8.1.1 Overview 34](#_Toc481705183)

[8.1.2 Introduction 34](#_Toc481705184)

[8.1.3 Signals 34](#_Toc481705185)

[8.1.4 Submodules 35](#_Toc481705186)

[9 Retire Unit 36](#_Toc481705187)

[9.1 Overview 36](#_Toc481705188)

[9.1.1 Introduction 36](#_Toc481705189)

[9.1.2 Signals 36](#_Toc481705190)

[9.1.3 Submodules 37](#_Toc481705191)

[10 Interrupt 38](#_Toc481705192)

[11 Debug Unit 39](#_Toc481705193)

[12 Coprocessor 40](#_Toc481705194)

**Table of Figures**

[Figure 1‑1 Block Diagram of Ace21064 Processor 7](#_Toc481705195)

[Figure 2‑1 Instruction Cache Hierarchy Overview 9](#_Toc481705196)

[Figure 2‑2 ace\_icache block diagram 10](#_Toc481705197)

[Figure 3‑1 BHT Port Timing 17](#_Toc481705198)

**Table of Tables**

[Table 3‑1 fetch unit port signals 12](#_Toc481705199)

[Table 3‑2 ace\_fetch sub-modules 13](#_Toc481705200)

[Table 3‑3 nvmc\_biu ports 15](#_Toc481705201)

[Table 3‑4 BHT Port List 16](#_Toc481705202)

[Table 3‑5 BHT Port List 18](#_Toc481705203)

[Table 4‑1 decode unit port signals 21](#_Toc481705204)

[Table 4‑2 ace\_decode sub-modules 22](#_Toc481705205)

[Table 5‑1 IRU port signals 26](#_Toc481705206)

[Table 5‑2 ace\_rename sub-modules 27](#_Toc481705207)

[Table 6‑1 decode unit port signals 29](#_Toc481705208)

[Table 6‑2 ace\_decode sub-modules 30](#_Toc481705209)

[Table 7‑1 decode unit port signals 32](#_Toc481705210)

[Table 7‑2 ace\_decode sub-modules 33](#_Toc481705211)

[Table 8‑1 decode unit port signals 34](#_Toc481705212)

[Table 8‑2 ace\_decode sub-modules 35](#_Toc481705213)

[Table 9‑1 decode unit port signals 36](#_Toc481705214)

[Table 9‑2 ace\_decode sub-modules 37](#_Toc481705215)

# Ace 21064 Core Basics

## Overview

ACE 21064 is a RISCV ISA based quard-issue superscalar processor core, which has 12 stage integer pipeline

Figure ‑ Block Diagram of Ace21064 Processor

## Features

* Quard-issue out-of-order 12-stage integer pipeline
* 64bit virtual address, 40bits physical address
* 8KB Virtually indexed physically tagged 2-way set associative instruction cache
* 4KB per-page in memory manager
* G-Share and Pap Tournament Hybird branch predictor
* 8 entry return address stack(RAS)
* 32 entry instruction buffer
* 4-instruction can be decoded in one CPU cycle
* 4-wide rename from 80 physical registers file, which is 65-bits each, renaming in unified method
* the physical register file can be accessed from 11 read port or 7 write port
* speculative and architectural rename maps, with respective RAT and FREELIST
* Checkpoint-based mechanism for fast recovery from branch misprediction
* Store Set based memory dependence prediction
* 32-entry scheduler used in issue stage
* speculative wakeup mechanism
* instruction replay
* centralized and distributed combined reservation station is used for efficiency
* Non-data-capture architecture is used in register read phase
* the reservation station is designed in compressed mode
* 2 simple ALU is used for add sub shift etc.
* 1 complex ALU for multiply and divide operation
* 1 branch unit designed for branch instructions
* 2 address generation unit, designed for load and store operation
* 16-entry load and store queue, to keep data before data cache access
* Dual-port 2-way associative l1 data cache is designed with 8 interleaved banks 4KB each bank
* 64-entry reorder buffer is used to support 8 wide retirement

## Components

## Pipeline architecture of Ace21064



# Level 1 Instruction Cache

## Overview

Level 1 instruction cache of Ace21064 is a two way-associative, virtually indexed and physically tagged cache, each data entry has 32 bytes, 128 cache lines in each way, 40bits physical address is designed for temporary, so 28bits higher address range is kept in tag array of each cache line, Total size of l1icache is 8KB, and memory is managed in 4KB page size.

### Introduction

In order to reduce the hit time of level 1 cache, virtually indexed, physically tagged cache architecture is used, as widely used in modern processor. The overview of instruction cache system of Ace21064 is showing below:



Figure ‑ Instruction Cache Hierarchy Overview

Ace21064 is 64bits architecture, and physical address is remained to 40bits for current design. Totally 1TB address range can be accessed, the memory system is managed in 4KB page size, so the low 12bits of virtual address is used as page offset, due to the two-way associative cache organization the total cache capability is 8KB large. To support the 4-issue pipeline, the fetch stage is designed in 8 instructions per-fetch, so the instruction cache is designed to 8 words per-entry for design convenient. Totally 32bytes needs 5 bits to index each byte in a cache entry, named block offset. The width of index segment can be calculate following the formula:

Or following the page width, 7 bits are left for cache line index, so there are 128 cache line in each way.

According the width of physical address, 28bits need to be kept in tag array of icache, which will be compared with the hit-value of iTLB to determine the fetch operation is hit or not.

### Architecture

Figure ‑ ace\_icache block diagram

### Signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Common Interface | | | |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_s0\_i | input | 64 | program counter in pipeline stage 0 |
| pc\_s1\_i | input | 64 | program counter in pipeline stage 1 |
| Interface Between MMU and l1icache | | | |
| itlb\_phystag0\_i | input | 28 | physical tag from itlb0 |
| itlb\_phystag1\_i | input | 28 | physical tag from itlb1 |
| itlb\_miss0\_i | input | 1 | itlb miss flag |
| itlb\_miss1\_i | input | 1 | itlb miss flag |
| l1icache output | | | |
| ic\_stall\_s0\_o | output | 1 | l1icache stall flag |
| ic\_inst\_s1\_o | output | 256 | l1icache data out |
|  | | | |
|  |  |  |  |

### Submodules

### Timing

## Instruction Alignment Unit

### Introduction

### Architecture

### Signals

### Submodules

### Timing

# Instruction Fetch Unit (IFU)

## Overview

### Introduction

Instruction fetch is responsible for providing a continuous instruction stream to the rest of the pipeline, Ace21064’s fetch stage achieves a fetch bandwidth of 8 instructions from level 1 instruction cache. And a dynamic branch predictor to speculate on the outcome of a branch instruction. The branch prediction mechanism is composed of four major hardware structures: branch target buffer (BTB), branch prediction stage0 (BPD0), branch prediction stage1 (BPD1), return address stack (RAS), and branch ordering buffer (BOB) etc. The block diagram is shown as below.

### Signals

Table ‑ fetch unit port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_f0 | input | 64 | program counter in fetch stage 0 |
| pc\_f1 | input | 64 | program counter in fetch stage 1 |
| signals from retire stage | | | |
| flush\_rt\_i | input | 1 | flush signal from retire stage |
| flush\_pc\_rt\_i | input | 64 | flush PC from retire stage |
| brcond\_vld\_rt\_i | input | 1 | condition branch retired flag |
| brindir\_vld\_rt\_i | input | 1 | indirect branch retired flag |
| brdir\_rt\_i | input | 1 | branch direction from retire stage |
| Signals from iCache | | | |
| inst\_align\_i | input | 256 | instruction fetch group1 |
| icache\_stall\_i | input | 1 | indicate the stall status of icache |
| Temporary | | | |
| inst\_q\_full\_i | input | 1 | come from instruction queue, which indicates the full status |
| Output signal for pc generator | | | |
| branch\_pc\_o | output | 64 | branch target pc from BTB or RAS |
| override\_pc\_o | output | 64 | override pc value from fetch1 stage |
| override\_vld\_o | output | 1 | override valid flag from fetch1 stage |
|  |  |  |  |
| Output signal for instruction queue | | | |
| inst0\_vld\_r | output | 1 | instruction0 valid signal registered from fetch stage1 |
| inst1\_vld\_r | output | 1 | instruction1 valid signal registered from fetch stage1 |
| inst2\_vld\_r | output | 1 | instruction2 valid signal registered from fetch stage1 |
| inst3\_vld\_r | output | 1 | instruction3 valid signal registered from fetch stage1 |
| inst4\_vld\_r | output | 1 | instruction4 valid signal registered from fetch stage1 |
| inst5\_vld\_r | output | 1 | instruction5 valid signal registered from fetch stage1 |
| inst6\_vld\_r | output | 1 | instruction6 valid signal registered from fetch stage1 |
| inst7\_vld\_r | output | 1 | instruction7 valid signal registered from fetch stage1 |
| inst0\_ali\_r | output | 32 | aligned instruction0 registered from fetch stage1 |
| inst1\_ali\_r | output | 32 | aligned instruction1 registered from fetch stage1 |
| inst2\_ali\_r | output | 32 | aligned instruction2 registered from fetch stage1 |
| inst3\_ali\_r | output | 32 | aligned instruction3 registered from fetch stage1 |
| inst4\_ali\_r | output | 32 | aligned instruction4 registered from fetch stage1 |
| inst5\_ali\_r | output | 32 | aligned instruction5 registered from fetch stage1 |
| inst6\_ali\_r | output | 32 | aligned instruction6 registered from fetch stage1 |
| inst7\_ali\_r | output | 32 | aligned instruction7 registered from fetch stage1 |
| inst0\_pc\_f1\_r | output | 64 | pc value of aligned instruction0 registered from fetch stage1 |
| inst1\_pc\_f1\_r | output | 64 | pc value of aligned instruction1 registered from fetch stage1 |
| inst2\_pc\_f1\_r | output | 64 | pc value of aligned instruction2 registered from fetch stage1 |
| inst3\_pc\_f1\_r | output | 64 | pc value of aligned instruction3 registered from fetch stage1 |
| inst4\_pc\_f1\_r | output | 64 | pc value of aligned instruction4 registered from fetch stage1 |
| inst5\_pc\_f1\_r | output | 64 | pc value of aligned instruction5 registered from fetch stage1 |
| inst6\_pc\_f1\_r | output | 64 | pc value of aligned instruction6 registered from fetch stage1 |
| inst7\_pc\_f1\_r | output | 64 | pc value of aligned instruction7 registered from fetch stage1 |

### Submodules

Table ‑ ace\_fetch sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
| BRDEC | pre-decode logic for branch identification |
| BPD0 | branch prediction stage0 |
| BPD1 | branch prediction stage1 |
| BTB | branch target buffer |
| BOB | branch order buffer |
| RAS | return address stack |
|  |  |

## Branch Prediction Unit

### Introduction

Ace 21064 use a Tournament predictor

### Architecture



### Ports

Table ‑ nvmc\_biu ports

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| hclk | Input | 1 | Clock |
| hrst\_b | Input | 1 | Reset |
| Memory access bus interface | | | |
| hsel | Input | 1 | Memory access bus interface |
| htrans | Input | 2 | Memory access bus interface |
|  |  |  |  |
|  |  |  |  |
| Slave 1 access bus interface | | | |
| s1\_hsel | Output | 1 | EFC access bus interface |
| s1\_htrans | Output | 2 | EFC access bus interface |
|  |  |  |  |
| Slave 2 access bus interface | | | |
| s2\_hsel | Output | 1 | ROM1 access bus interface |
| s2\_htrans | Output | 2 | ROM1 access bus interface |

### Timing

### Branch History Table (BHT)

BHT in Ace21064 has 10 bits wide, which can record ten branch taken history. And has 1024 entries for different branches.

Read BHT :

Read Address: (Read Index) there are 1K entries in design, so the read index width should be 10 bits from current PC. As we know the RISCV ISA is 32bit width, so the bits [1:0] is reserved. bht\_rd\_index\_i[9:0] comes from cur\_pc\_i[11:2].

Read Data: when we find the entry which the cur\_pc has indexed, read the data out for PHT index. We rename it as bht\_br\_hist\_o[9:0].

Write BHT:

Write Address: (Write Index) we update BHT when the branch instruction is committed (after execute stage, the branch direction is confirmed) we named reference pc as confirmed pc. So bht\_wt\_index\_i[9:0] comes from cm\_pc\_i[11:2]

Write Data: As the definition of BHT (a shifter), the write data is the branch direction of the confirmed pc instruction, (bht\_cm\_brdir\_i)

Write enable: except address and data we need to know when to update the BHT entry, so write enable signal is needed. As we know the BHT entry can only be update when the branch is confirmed for the correct branch history record. (bht\_cm\_brdir\_se\_i) (fixme: when to update the BHT [SuperScalar RISC Processor Design p118])

Table ‑ BHT Port List

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_b | Input | 1 | Reset |
| BHT Read Interface | | | |
| bht\_rd\_index\_i | Input | 10 | BHT read index, part of current pc |
| bht\_br\_hist\_o | Output | 10 | Branch history, BHT read data |
| BHT Write Interface | | | |
| bht\_wt\_index\_i | Input | 10 | BHT write index, part of confirmed branch instruction’PC |
| bht\_cm\_brdir\_i | Input | 1 | confirmed branch instruction’s direction |
| bht\_cm\_brdir\_se\_i | Input | 1 | confirmed branch instruction direction shift enable |



Figure ‑ BHT Port Timing

### Pattern History Table (PHT)

BHT in Ace21064 has 10 bits wide, which can record ten branch taken history. And has 1024 entries for different branches.

Read BHT :

Read Address: (Read Index) there are 1K entries in design, so the read index width should be 10 bits from current PC. As we know the RISCV ISA is 32bit width, so the bits [1:0] is reserved. bht\_rd\_index\_i[9:0] comes from cur\_pc\_i[11:2].

Read Data: when we find the entry which the cur\_pc has indexed, read the data out for PHT index. We rename it as bht\_br\_hist\_o[9:0].

Write BHT:

Write Address: (Write Index) we update BHT when the branch instruction is committed (after execute stage, the branch direction is confirmed) we named reference pc as confirmed pc. So bht\_wt\_index\_i[9:0] comes from cm\_pc\_i[11:2]

Write Data: As the definition of BHT (a shifter), the write data is the branch direction of the confirmed pc instruction, (bht\_cm\_brdir\_i)

Write enable: except address and data we need to know when to update the BHT entry, so write enable signal is needed. As we know the BHT entry can only be update when the branch is confirmed for the correct branch history record. (bht\_cm\_brdir\_se\_i) (fixme: when to update the BHT [SuperScalar RISC Processor Design p118])

Table ‑ BHT Port List

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_b | Input | 1 | Reset |
| BHT Read Interface | | | |
| bht\_rd\_index\_i | Input | 10 | BHT read index, part of current pc |
| bht\_br\_hist\_o | Output | 10 | Branch history, BHT read data |
| BHT Write Interface | | | |
| bht\_wt\_index\_i | Input | 10 | BHT write index, part of confirmed branch instruction’PC |
| bht\_cm\_brdir\_i | Input | 1 | confirmed branch instruction’s direction |
| bht\_cm\_brdir\_se\_i | Input | 1 | confirmed branch instruction direction shift enable |

## Return Address Stack (RAS)

## Branch Target Buffer (BTB)

### Introduction

4-Way associative branch target buffer is realized in Ace 21064 processor

### BTB Way

### Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_n | Input | 1 | Reset |
| Memory access bus interface | | | |
| hsel | Input | 1 | Memory access bus interface |
| htrans | Input | 2 | Memory access bus interface |
|  |  |  |  |
|  |  |  |  |
| Slave 1 access bus interface | | | |
| s1\_hsel | Output | 1 | EFC access bus interface |
| s1\_htrans | Output | 2 | EFC access bus interface |
|  |  |  |  |
| Slave 2 access bus interface | | | |
| s2\_hsel | Output | 1 | ROM1 access bus interface |
| s2\_htrans | Output | 2 | ROM1 access bus interface |

## PC Generator



# Instruction Decoder Unit(IDU)

## Overview

### Introduction

Decode stage in Ace21064 processor contains two physical pipeline stages(two cycles), first is an instruction buffer, which receives up to 8 instructions from IFU, these instructions are write into a circular buffer with tail pointer, and only 4 instructions are read from the head pointer to the second stage of instruction decoder, feed 4 wide decoder, 4 instructions can be decoded in one cycle.

.

### Signals

Table ‑ decode unit port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_f0 | input | 64 | program counter in fetch stage 0 |
| pc\_f1 | input | 64 | program counter in fetch stage 1 |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Submodules

Table ‑ ace\_decode sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
| inst\_buf | instruction buffer, the fetch stage will fetch 8 instructions per cycle, but decode stage can only decode 4 instructions one time |
| decoder\_0 | decode unit 0 |
| decoder\_1 | decode unit 1 |
| decoder\_2 | decode unit 2 |
| decoder\_3 | decode unit 3 |

## Instruction Buffer

### Introduction

Instruction buffer receives up to 8 instructions from IFU, these instructions are write into a circular buffer with tail pointer, and only 4 instructions are read from the head pointer to the second stage of instruction decoder, the real decode stage.

The instruction buffer allows instruction fetching, even the rest of frontend pipeline is stalled because of resource limitation, and because of the existence of this buffer, the decode, rename, and dispatch stage can always be fed with a fix number of instruction, 4 instruction in current design.

### Architecture

### Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_n | Input | 1 | Reset |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |

## Instruction Decoder

### Introduction

The instruction decode logic is the most clear module, due to the convenient definition of RISC-V ISA, more information of RISC-V ISA refer to doc[riscv-spec-v2.1.pdf].

Currently, Ace21064 only implements integer instructions, and we intend to extend the design for floating-point instructions in next step.

### Architecture

### Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_n | Input | 1 | Reset |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |

# Instruction Renaming Unit

## Overview

Register renaming removes the false dependencies among instructions which are artifacts of limited architectural registers. The data dependencies of a dynamic instruction stream can be classed as:

True dependency, which the source register of a younger instruction depends on the outcome of another older instruction in the instruction stream, this dependency also named read after write dependency (RAW)

Output dependency, where the destination register of a younger instruction is the same as the destination register of another older instruction in the instruction stream, this dependency also called write after write dependency (WAW).

Anti dependency, where the destination register of a younger instruction is the same as the source register of another older instruction in the instruction stream, this dependency also named as write after read dependency (WAR).

WAW and WAR sometimes also referred as false dependencies, register renaming eliminates false dependencies by mapping architectural destination register of each in-flight instruction to a unique physical register

### Introduction

Current design implements 4 wide rename from 80 physical registers, both speculative and architectural rename maps maintained, a circular FIFO is implemented as speculative free list (SPEC\_FREELIST), contains the unused physical registers. An unused physical register is popped by the speculative free list to be used as a replacement of the architectural destination register of instruction. A register alias table (RAT) maintains the physical registers to which architectural registers are currently mapped. Accordingly, each architectural source register of the instruction is renamed to a physical source register by looking up RAT.

Memory dependence prediction using store sets.

### Signals

Table ‑ IRU port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
|  |  |  |  |
|  |  |  |  |
|  | | | |

### Submodules

Table ‑ ace\_rename sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
| spec\_rat | speculative RAT with 8 read ports(four-wide renaming each instruction has two source register in worst case), 4 write ports(four-wide renaming, each instruction has one destination register in worst case). |
| spec\_freelist | speculative FreeList with 4 read ports(four-wide renaming per cycle), 8 write ports (eight wide retire per cycle). |
| mdp\_ssit | memory dependence predictor store set ID table |
| mdp\_ssit\_dpd\_chk | identifies intra instruction bundle dependencies for store set IDs |
| mdp\_lfst | last fetched store table for memory dependence prediction |
| mdp\_dpd\_chk | does dependence check between the 4 instructions currently in rename stage0 |
|  |  |
|  |  |
|  |  |

## Instructions Renaming Unit

### Introduction

For a 4-way renaming stage, there are 4 instructions should be renamed in one cycle, so there are eight source register and four destination register in worst case. And SPEC\_FREELIST and RAT (pop four destination register, read eight source register) are accessed in the same cycle,

NOTE: if there are multiple producers of the same architectural registers in the rename group, only the youngest producer updates the RAT.

In current design the RAT is implemented with SRAM, so the RAT is also called SRAM based RAT (sRAT, contrast with CAM based RAT, cRAT). According these the sRAT should designed with 8 read ports, and 4 write ports. Considering the speculative execution in branch instruction, if the misprediction occurs, the RAT need to be recover from Architectural RAT, implemented in retire stage.(will be introduced in following chapter), so another write port needed to be added. sRAT needs 8 read ports and 5 write ports FOR NOW. consider checkpoint.

What’s more, before we update RAT, we should keep the old data in sRAT, for two reasons: first, when current instruction retire corresponding physical register should be set as free (in freelist); second, if exception triggered or misprediction take place, the pipeline should be flushed, and we need to recover RAT, so we need another read port to devolve RAT data.

### Architecture

### Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_n | Input | 1 | Reset |
|  | | | |

## Memory dependency predictor

### Introduction

### Architecture

### Ports

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| Global interface |  |  |  |
| clk | Input | 1 | Clock |
| rst\_n | Input | 1 | Reset |
|  | | | |

# Instruction Schedule Unit(ISU)

## Overview

### Introduction

32-entry scheduler.

Speculative wakeup

One cycle execute instruction: wakeup and select in one cycle, to make RAW dependency instruction can be executed back to back

Multiply cycle execute instruction: delayed wakeup strategy

Instruction replay(replay queue based replay is used)

Several function unit share the same reservation station, combined with centralized and distributed reservation station are used.

Non-data-capture architecture(read source data after instruction was selected in reservation station) is used

Compressed reservation station is choosed.

### Signals

Table ‑ decode unit port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_f0 | input | 64 | program counter in fetch stage 0 |
| pc\_f1 | input | 64 | program counter in fetch stage 1 |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Submodules

Table ‑ ace\_decode sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Execution Unit

## Overview

### Introduction

.

### Signals

Table ‑ decode unit port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_f0 | input | 64 | program counter in fetch stage 0 |
| pc\_f1 | input | 64 | program counter in fetch stage 1 |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Submodules

Table ‑ ace\_decode sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Level 1 data Cache

### Overview

### Introduction

.

### Signals

Table 8‑1 decode unit port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_f0 | input | 64 | program counter in fetch stage 0 |
| pc\_f1 | input | 64 | program counter in fetch stage 1 |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Submodules

Table 8‑2 ace\_decode sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Retire Unit

## Overview

### Introduction

64-entry reorder buffer, with eight-wide retire

Missprediction

Exception

Interrupt

Store operation

storebuffer

### Signals

Table 9‑1 decode unit port signals

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Direction | Width | Description |
| clock | input | 1 | system clock |
| reset\_n | input | 1 | system reset low active |
| pc\_f0 | input | 64 | program counter in fetch stage 0 |
| pc\_f1 | input | 64 | program counter in fetch stage 1 |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

### Submodules

Table 9‑2 ace\_decode sub-modules

|  |  |
| --- | --- |
| Module Name | Description |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

# Interrupt

# Debug Unit

# Coprocessor

Power management

Performance Monitors