The Tale of Alpha Series Microprocessor

Yingkun Zhou
zhouyingkun15@mails.ucas.ac.cn

November 25, 2018

In the current stage, I want to focus the tech of designing high performance uniprocessor based on my latest knowledge, as a senior undergraduate student. Though I know that a computer system nowadays is far away from a powerful uniprocessor. It means that we not only need to fully explore parallelism and thus introduce multiprocessor, but also means that but also need to develop full-featured software environment which can run in our ISA, which is considered as the most important aspect. Intel and DEC are the pair of such example. In 1990s, DEC has the most powerful microprocessor–Alpha series at that time, with peak frequency up to 600MHz, 64 bits architecture, 4-issue, 80-instruction execution at same time, while Intel's fastest microprocessor–Pentium II only has 300MHz peak frequency with 32 bits, poorly designed architecture and the worst ISA. Ironically, Intel survived while DEC dead, quite funny history. But, anyway, as a milestone in the microprocessor history, the tech reflected in the Alpha design still worth learning, especially for me, who wants to design self high performance CPU (based on RISC-V ISA maybe).

1.low digit circuit aspect[1]

There are there generation of Alpha microprocessors. Following is the features.

- 21064: 1.68×10^6 transistors, 200MHz, .75μm n-well CMOS process, 16 gate delays/cycle, 3.3V, 30W
- 21164: 9.3×10^6 transistors, 300MHz, .50μm n-well CMOS process, 14 gate delays/cycle, 3.3V, 50W
- 21264: 1.52×10^7 transistors, 600MHz, .35μm n-well CMOS process, 12 gate delays/cycle, 2.2V, 72W

Some features of the Alpha ISA:

i 64-bit lw/sw RISC arch

ii Fix-length instructions

iii minimal instruction ordering constraints

iv 64-bit manipulation allow for straightforward instruction decode (**confused**)

v does not contain condition codes (like x86), branch delay slots (like MIPS), adaptations from existing 32-bit arch (**bad for Compatibility**)

[1] reading note from *High-performance Microprocessor*

CPU inner arch impl detail:

- 21064: fully pipelined, in-order execution, 2-issue, one pipelined integer execution unit (**one or two cycles except for multiplies which are not pipelined**), one pipelined floating point execution unit (**6 cycles for all instructions except for divides**). 8KB Icache, 8KB Dcache
- 21164: quad-issue, in-order execution; contains 2 pipelined integer execution units (**one cycle for all integer instructions and roughly halved for all MUL instructions**), 2 pipelined floating-point execution units (separate add and multiply pipelines, 4-cycle latency); L1-level cache was changed to **nonblocking**; L2-level 96KB unified I and D cache.
- 21264: 6-way-issue and out-of-order execution; 4 integer execution units and 2 floating-point execution units; L1-Icache and L1-Dcache are both 64KB (eliminate on-chip L2 cache); reduce the divide latency by 50% (extends to include square root and multimedia instructions)

The III part is confusing especially *definition of Design Rules* thanks to my poor knowledge about the CMOS process field.

Circuit style highlight of Alpha series (**impl with a wide range of circuit styles including conventional complementary CMOS logic, single- and dual-rail dynamic logic, cascode logic pass transistor logic, and ratioed static logic**):

i full-custom circuit design methodologies.

ii Dynamic circuits: widely used in both data path and random control structures. The advantages are following:

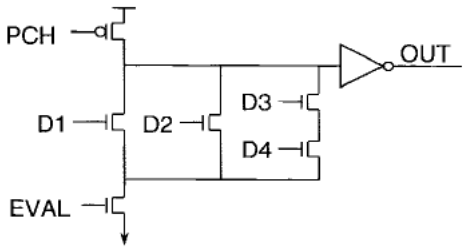


Fig. 6. Dynamic logic example.

(a) wide OR, no need many levels of complementary logic

(b) eliminating the PMOS transistor network reduces both the gate fan-int and fan-out capacitance.

(c) the switching point of the dynamic gate is set by the NOMS device threshold voltage. (if the timing of the inputs is such that they are not asserted until after the precharge clock has been deasserted, there in no crossover current during the output transition) **a bit of confused**

(d) Removing the PMOS transistor network reduces the layout area, which results in lower interconnect capacitance, further increasing the speed of the circuit.

iii dual-rail cascode logic, used in both data path and random control logic areas. The advantages are following:

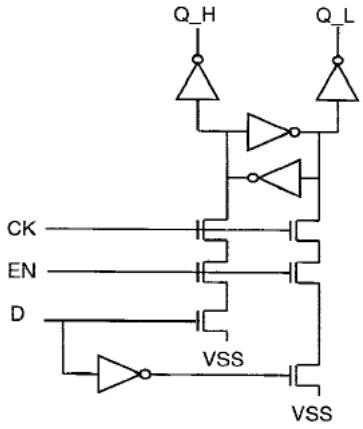


Fig. 7. Cascode logic example.

(a) the fan-in and fan-out capacitance are both lower, thus reducing delay.

(b) Large complex functions such as multiplexers and XOR gates can be easily implemented in a single cascode gate with both true and complement outputs.

(c) a latch function can easily be constructed with addition of one pair of transistors.

Power Dissipation and Supply Distribution part is **quite confusing**. Again based on my poor knowledge cannot understand the analysis why to add aluminum interconnect layer(M3 lines), M4, and two thick low-resistance aluminum reference planes step by step to satisfy the limitation of power. Just put the graph here in case of need for discussing.

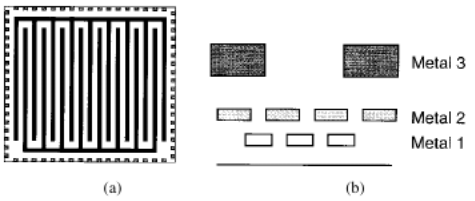


Fig. 10. (a) 21064 metal layers and (b) power distribution.

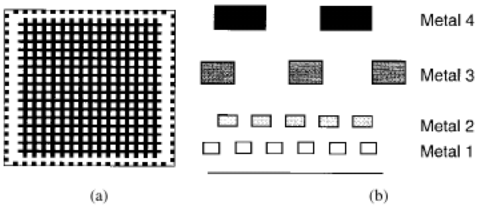


Fig. 11. (a) 21164 metal layers and (b) power distribution.

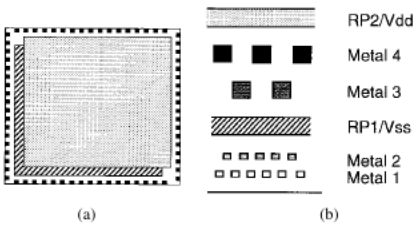


Fig. 12. (a) 21264 metal layers and (b) power distribution.

Clock Distribution

The motivation of this part is that the high frequencies of the Alpha microprocessors have required the generation and distribution of a very high-quality clock signal and the use of fast (low-latency) latches.

Here is the possible impact of clock system on the performance of the microprocessor:

- Uncertainties in clock edges resulting form power supply noise, process variation and interconnect RC delay lower the maximum clock frequency of the microprocessor.
- slow clock edges introduce uncertainties in latch timing which further limit performance and can lead to functional failures due to latch race-through.

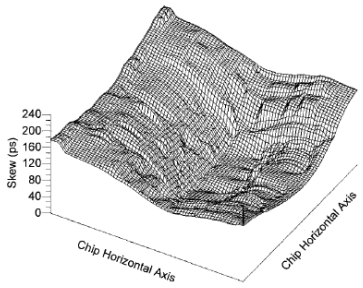


Fig. 14. 21064 clock skew.

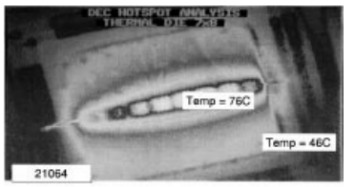


Fig. 16. 21064 thermal image.

(a) clock driver location (b) clock skew (c) thermal imag

Figure 1: Alpha 21064 clock system analysis (a mouse with teeth, see right most graph)

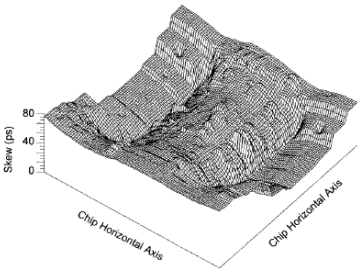


Fig. 17. 21164 clock skew.

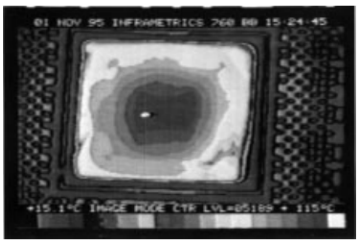


Fig. 18. 21164 thermal image.

(a) clock driver location (b) clock skew (c) thermal imag

Figure 2: Alpha 21164 clock system analysis

From above figures we can see that the nearer clock driver, the lower latency skew and the higher temperature.

And below is the approximate power breakdown of the 21064 and the 21164

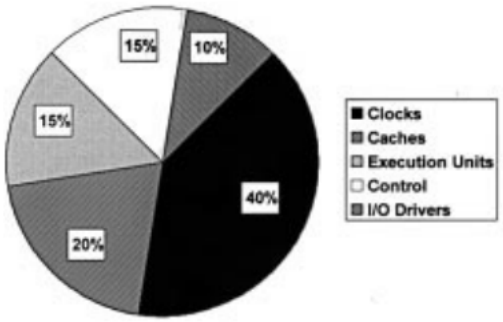


Fig. 15. Approximate power breakdown of the 21064 and the 21164.

It tells us that the clock driver is the main power consumer, consuming 40% of the chip power.

And as more aggressive circuit techniques and complex micro-architecture features were implemented in the 21264, power consumption became a major concern in designing the clocking system. Thus, new tech was introduced–GCLK to reduce clock grid RC delay and distribute clock power.

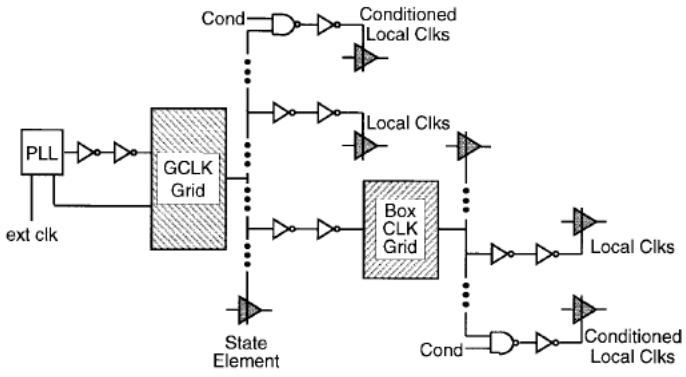


Fig. 19. 21264 clock hierarchy.

So what is GCLK?

GCLK is the root of a hierarchy of thousands of buffered and conditioned local clocks used across the chip as shown in the above picture. There are 2 advantages and 1 dis.

- adv: conditioning the local clocks saves power.
- adv: circuit designers can take advantage of multiple clocks (for example, a phase path can).
- disad: significantly complicates race and timing verification.
- notice: using local buffering significantly lowers the GCLK load, which reduces GCLK skew.

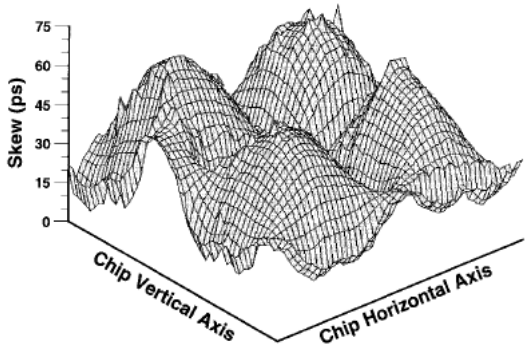


Fig. 20. 21264 GCLK skew.

Next part of the paper talks about latch design. Recall my Digital circuit course, I've learned several kinds of latch and flip-flops. Realize it is really a significant element of the microprocessor circuit design strategy. In other word, no latch and flip-flops, no timing logic, no CPU! And as the paper mentioned "In order to ensure proper operation across all operating conditions, clock and latch circuits cannot designed independently." In fact, each generation of the Alpha microprocessor has utilized latches with improved characteristic combined with the improvements to the clock distribution networks previously described.

Thus, the motivation of making low-latency latch design is for high speed. Besides, other primary goals in latch design are minimal area and clock loading, low power dissipation, and low setup and hold times.

Some highlight I've ever seen in general and in each generation of the Alpha microprocessor.

i **In general**
The capability of embedding a logic function directly in the latch helps reduce the number of gate delays per cycle. (**Confused**)

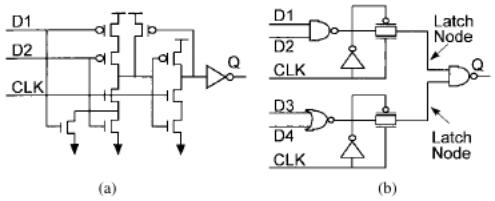
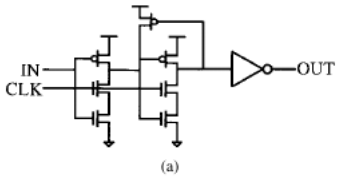


Fig. 22. Embedding logic into latches: (a) 21064 Function Latch: one level of logic; (b) 21164 Function Latch: two levels of logic.

the paper says that in some instances, both inverters were replaced with logic gates, further reducing the latching families are show in above graph. It can be seen that the cost of latching in the 21164 could be reduced to a minimum of one pass transistor.

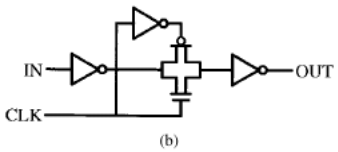
ii **21064**
Digital's first microprocessor to use a two-phase, single wire-clocking scheme (this was radical departure from the four-phase scheme that allowed for free design in previous versions of Digital's microprocessors).

Reducing the chance of data race-through using a variation of the true single-phase clocked (TSPC) level-sensitive latch (an example see below graph).



feature: using the unbuffered main clock directly, significantly increasing race immunity. In addition, the first stage of this latch can incorporate a simple logic function.

iii **21164**
using dynastic CMOS transmission gate latch. (an basic example see below graph)



feature: requiring true and complementary clock signals, one of which is generated locally. The clock buffer for each latch type was custom designed so that its delay and edge rate characteristics could be tightly controlled. The additional buffer delays the clocking of the latch by one gate delay after the global clock transitions. (However, since the preceding latch opens with the global clock transition, the possibility of latch race-through is significantly increased. In order to minimize the possibility of data race-through with the use of these latches, at least one minimum logic delay element was required between all latches.)

iv **21264**
Requiring a static latch design and using a family of edge-triggered flip-flops, based on the dynamic flip-flop show in following graph.

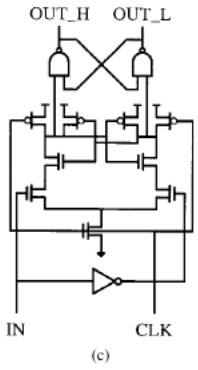


Fig. 21. Latches.

It was developed to simplify the timing and race issues that were exacerbated by the addition of conditional clocking. The change from level-sensitive to edge-triggered design techniques and the use of multiple clock buffers introduced a number of new timing issues to the design.

Critical path and race analysis

Goal: The capability of buffering and conditioning the main clock is possible as long as each circuit satisfies both its critical path and race requirements.

What does path and race analysis do?

it starts with the **identification of the common clock** initialing **both the receive and drive paths**, denoted R and D, Every critical path or race is defined by a **single common clock and a pair of receive and drive paths**. Here we say the two respective common clocks are GCLK & FCLK.

i Critical path analysis
verifies that the difference in delay between the drive path D plus the receiver setup time and the receive path R does ont exceed the phase or cycle time of the common clock. For worst case analysis, effects that minimize R and maximize D are considered.

ii Race analysis
Effects that maximize R and minimize D are considered. When the ratio of delay of the drive

path D to the receive path R, including hold time, is equal to 1, the circuit is on the verge of failing. Ratios (denoted by X) exceeding 1 imply a margin that may be used to account for effects not included in the analysis.

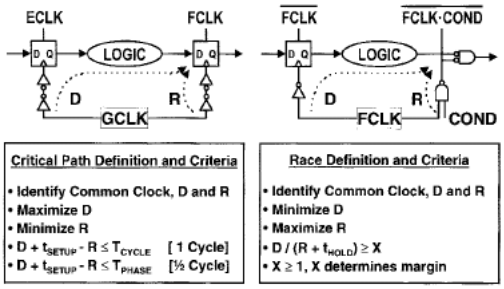


Fig. 23. Critical path and race analysis.

see above graph, the left example illustrates the use of two buffered section clocks, ECLK and FCLK, in a one-cycle path. The circuit on the right of the figure combines a buffered local clock and a conditioned local clock to define a one-phase path.

CAD tools and verification

Background: Commercially available EDA design systems and point CAD tools have very limited support for **Custom circuit techniques**. Therefore, Digital developed an extensive suite of in-house CAD tools to facilitate the design of custom VLSI microprocessors.

The internally developed CAD suite includes tools for

i schematic and layout entry

ii two-state and three-state logic simulation

iii RTL versus schematic equivalence checking

iv static timing analysis and race verification

v parameter and netlist extraction

vi electrical analysis and verification

There are two main parts

- Electrical Verification

  Covers all circuit issues that are not related to logic functionality such as timing behavior, electrical hazards and reliability. The primary goal is to verify that all circuits conform to the project design methodology.

  Some highlight of the tools

  - filter out all circuits that can easily be validated while identifying the small number of circuits that may have problems and require additional analysis.
  - perform over 100 unique electrical checks. and there are three around kinds
    - the major areas of focus are circuit topology violations, dynamic node checks, write-ability checks, latch checks, beta ratio checks, gate fan-in & fan-out restrictions, transistor size and stack height limitations, max/min edge rates and delays, and power consumption.

    - are applied to all circuit styles
    - are required for specific circuit types.

- Functional and Logical Verification

  Covers all phases of the design process from functional definition through manufacturing tests.

  A tow-state RTL behavioral model is the primary verification vehicle.

Future Challenges

i power consumption

ii the distribution of a chipwide timing reference with extremely fast edge rates

2.high logic "RTL" aspect[2]

The first part is mainly about custom circuit design techniques which is the CMOS level design and optimization. However, it is not very suitable for me to do some further works thanks to my poor base of digital circuit. Therefore, I'd like to focus on higher level which is called RTL level or functional logic level. And the design and optimization of this level is much easier to get hands dirty. Therefore, reading this paper is good to begin with my undergraduate thesis.

Alpha 21264 is a masterpiece in my eyes. It is elaborate designed and has many things to learn from.

This paper discusses two aspect, inner CPU core and outer cache and memory system.

- CPU core: A unique combination of high clock speeds and advanced microarchitectural techniques, including many forms of out-of-order and speculative execution
- memory system: high-bandwidth memory system that can quickly deliver data values to the execution core (including those without cache locality).

Architecture highlights

i is a superscalar microprocessor that can fetch and execute up to 4 instructions per cycle. It also features out-of-order execution.

ii employs speculative execution to maximize performance. In other words, when the 21264 predicts branch directions and speculatively executes down the predicted path. With more functional units and these dynamic execution techniques than 21164.

iii Its memory system also enables high performance levels. On-chip and offchip caches provide for very low latency data access. Additionally, the 21264 can service many parallel memory references to all caches in the hierarchy, as well as to the off-chip memory system.

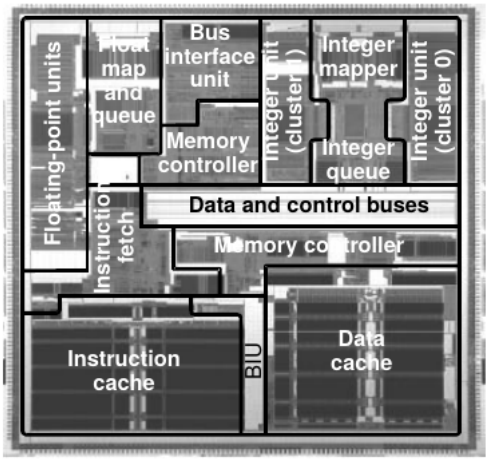


Figure 1 Alpha 21264 microprocessor die photo. BIU stands for bus interface unit.

Highlighting major sections of the 21264. We can see that on-chip cache is divided into two– Instruction cache and Dada cache. And it seems like there are the Bus interface unit has two parts, one is in the above and another is in the below. And Instruction fetch is in the middle which is very reasonable for high performance.

[2]reading note from *The Alpha 21264 Microprocessor*

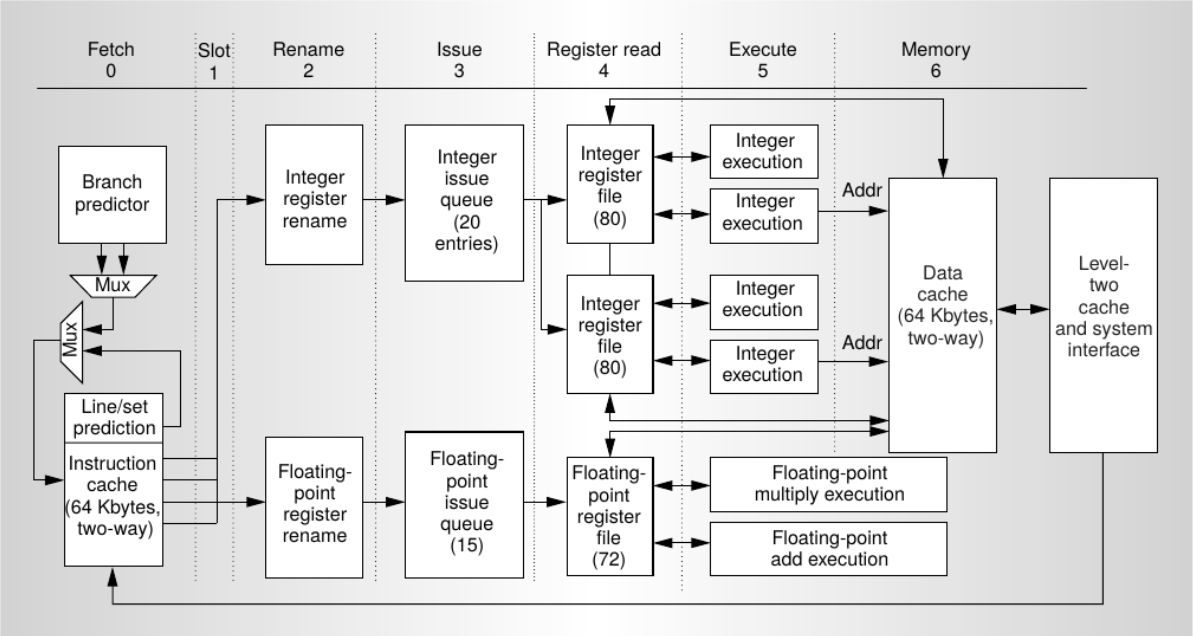


Figure 2. Stages of the Alpha 21264 instruction pipeline.

The pipeline is showed above, which has seven stages. One notable addition compared with 21164 is the map stage that renames registers to expose instruction parallelism–this addition is fundamental to the 21264's out-of-order techniques.

And now let's close look at each stage following with the paper.

- instruction pipeline–Fetch

  Two architectural techniques increase fetch efficiency: line and way prediction and branch prediction. The 21264 has a 64KB, two-way set-associative instruction cache offers much improved level-one hit rates compared to the 8KB, direct-mapped instruction cache in the Alpha 21164.

  i Line and way prediction

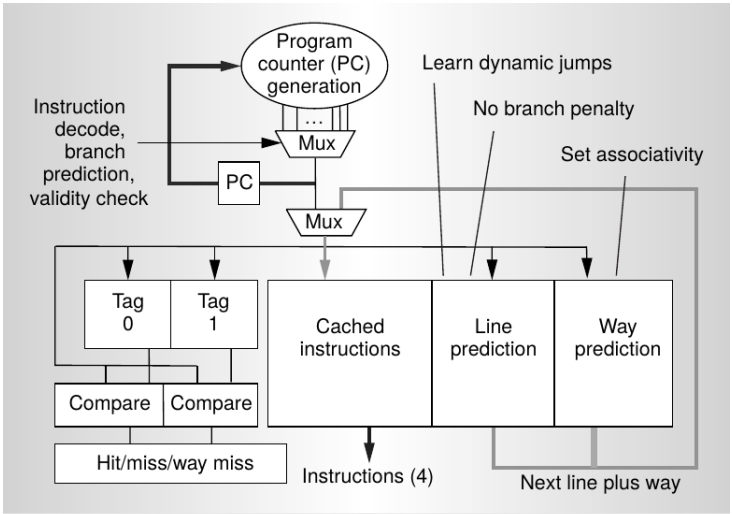


Figure 3. Alpha 21264 instruction fetch. The line and way prediction (wrap-around path on the right side) provides a fast instruction fetch path that avoids common fetch stalls when the predictions are correct.

Each 4-instruction fetch block includes a line and way prediction. This prediction indicates where to fetch the next block of 4 instructions, including which way–that is, which of the two choices allowed by 2-way associative cache. As an additional precaution, a 2-bit hysteresis counter associated with each fetch block eliminates overtraining–training occurs only when the current prediction has been in error multiple times (Line and way prediction is an important speed enhancement since the mispredict cost is low and line/way mispredictions are rare).

Besides, the fetch engine prefetches up to four 64-byte(or 16-instruction) cache lines to tolerate the additional latency.

ii Branch prediction

The 21164 could accept 20 in-flight instructions at most, but the 21264 can accept 80, offering many more parallelism opportunities. Thus, Branch prediction is more important to the 21264's efficiency than to previous microprocessors. Here are 3 considerations:

(a) the 7 cycles mispredict cost is slightly higher than previous generations.

(b) the instruction execution engine is faster than in previous generations.

(c) successful branch prediction can utilize the processor's speculative execution capabilities.

The 21264 implements a sophisticated tournament branch prediction scheme. The motivation is that the processor can adapt to dynamically choose the best method for each type of branch. Therefore, the scheme is designed to dynamically choose between two types of branch predictors

– using local history

– using global history

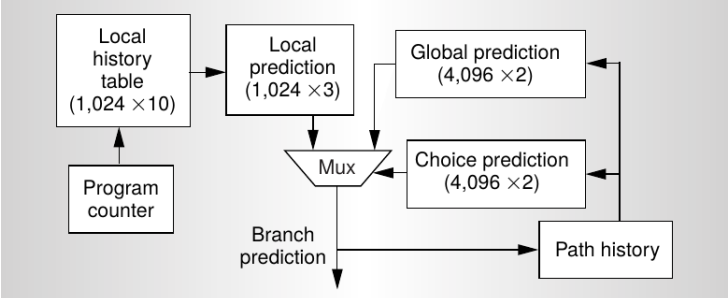


Figure 4. Block diagram of the 21264 tournament branch predictor. The local history prediction path is on the left; the global history prediction path and the chooser (choice prediction) are on the right.

in detailing the structure of the tournament branch predictor, shows

(a) the local history prediction path–through a 2-level structure–on the left. the first level holds 10 bits of branch pattern history for up to 1024 branches. This 10-bit pattern picks from one of 1024 prediction counters.

(b) the global predictor is a 4096-entry table of 2-bit saturating counters indexed by the path, or global, history of the last 12 branches.

(c) the choice prediction, or chooser, is also a 4096-entry table of 2-bit prediction counters indexed by the path history.

- Out-of-order execution

The out-of-order execution logic receives four-instructions every cycle, renames/remaps the registers to avoid unnecessary register dependencies, and queues the instructions until operands or functional units become available.

It dynamically issues up to 6 instructions every cycle–4 integer instructions and 2 floating point instructions.

i Register renaming

The 21264 speculatively allocates a register to each instruction with a register result. The register only becomes part of the user-visible (architectural) register state when the instruction retires/commits.

Register renaming also eliminates write-after-write and write-after-read register dependencies, but preserves all the read-after-write register dependencies that are necessary for correct computation.

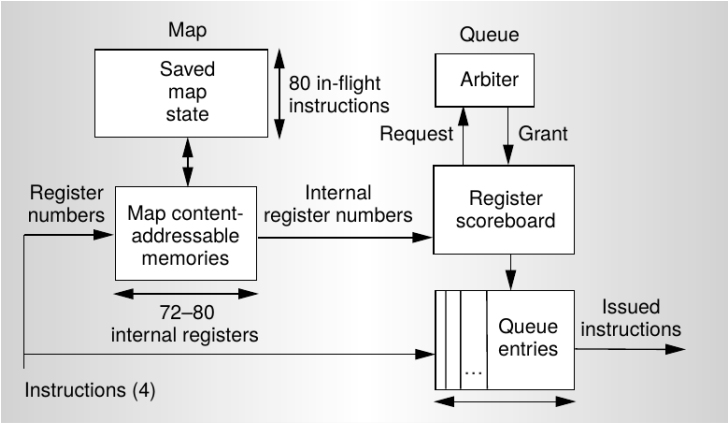


Figure 5. Block diagram of the 21264's map (register rename) and queue stages. The map stage renames programmer-visible register numbers to internal register numbers. The queue stage stores instructions until they are ready to issue. These structures are duplicated for integer and floating-point execution.

Register renaming is a content-addressable memory (CAM) operation for register sources together with a register allocation for the destination register.

Beyond the 31 integer and 31 floating-point user-visible (non-speculative) registers, an additional 41 integer and 41 floating-point registers are available to hold speculative results prior to instruction retirement.

ii Out-of-order issue queues

There are tow components. One is the queue logic itself which maintains two list of pending instructions in separate integer and floating-point queues. Anther is register scoreboards based on the internal register numbers.

How scoreboards works?

These scoreboards maintain the status of the internal register by tracking the progress of single-cycle, multiple-cycle, and variable-cycle (memory load) instructions. When functional-unit or load-data results become available, the scoreboard unit notifies all instructions in the queue that require the register value. These dependent instructions can issue as soon as the bypassed result becomes available from the functional unit or load.

The 20-entry integer queue can issue for 4 instructions and the 15-entry float-point queue can issue 2 instructions per cycle.

iii Instruction retire and exception handling

The 21264 implements a precise exception model using in order retiring, which means that the programmer does not see the effects of a younger instruction if an older instruction has an exception.

From issue until retire eligibility, how many cycles will take for each instruction? Well, it is up to the classes of instructions.

(a) Integer inst needs at least 4 cycles.

(b) Memory access inst needs at least 7 cycles.

(c) Floating-point inst needs at least 8 cycles.

(d) Branch/jump to subroutine inst needs at least 7 cycles.

Besides, the retire mechanism can retire at most 11 instructions in a single cycle, and it can sustain a rate of 8 per cycle (over short periods).

- Execution engine

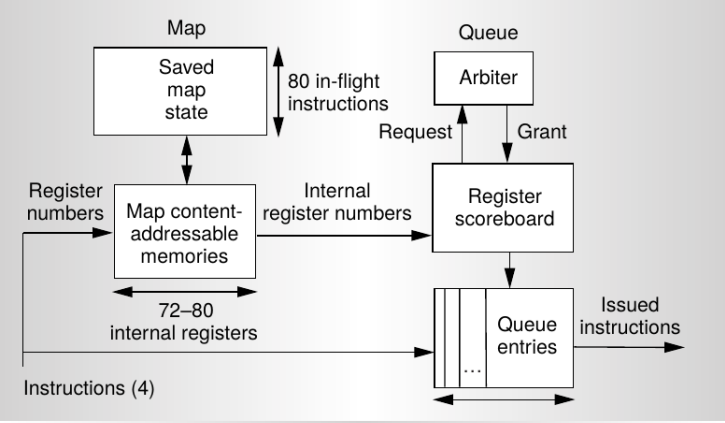


Figure 5. Block diagram of the 21264's map (register rename) and queue stages. The map stage renames programmer-visible register numbers to internal register numbers. The queue stage stores instructions until they are ready to issue. These structures are duplicated for integer and floating-point execution.

a bit of confused for the above graph

The clusters is a special design in my opinion. How it works? Two pipes access a single register file to form a cluster, and the two clusters combine to support four-way integer instruction execution. This clustering makes the design simpler and faster, although it costs an extra cycle of latency to broadcast results from an integer cluster to the other cluster. Other thing need to pay attention is that The upper pipelines from the two integer cluster are managed by the same issue queue arbiter, as are the tow lower pipelines. By the way, this part is still hard to understand until to implement by myself.

- Internal memory system (**this part is extremely important, need to more time to digest, and**

use these techniques in my own CPU memory system)

It is a high-band-width, low-latency memory system.

Here are several features:

i service up to 2 memory references from the integer execution pipes every cycle. These 2 references are out-of-order issues.

ii simultaneously tracks up to 32 in-flighting loads, 32 in-flighting stores, and 8 in-flighting (instruction or data) cache misses.

iii also has a 64KB, two-way set-associative data cache.

i Data path

The 21264 supports any combination of two loads/stores per cycle without conflict. How to achieve it? The data cache is double-pumped with two-ports and operates at twice the frequency of the processor clock – an important feature of the 21264's memory system.

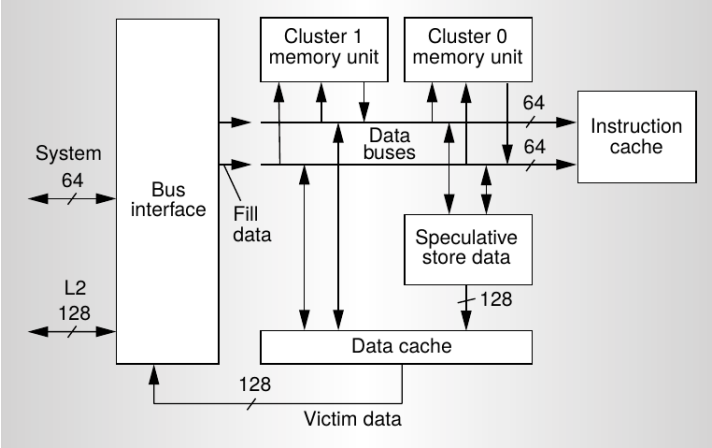


Figure 7. The 21264's internal memory system data paths.

The 2 64-bit data buses are the heart of the internal memory system. Let's consider two operations – load and store

- load: there are three sources can load data from:
  - the data cache
  - the speculative store data buffers
  - an external (system or L2) fill
- store: First transfer their data across buses into the speculative store buffer until the stores retire. Once they retire, the data is written (dumped) into the data cache on idle cache cycles. Each dump can write 128 bits into the cache since two stores can merge into one dump.

The optimization of load and store if we consider them together:

Stores can forward their data to subsequent loads while they reside in the speculative store data buffer. Load instructions compare their age and address against these pending stores. On a match, the appropriate store data form the data cache. In effect, the speculative store data buffer performs a memory-renaming function. From the perspective

of younger loads, it appears the stores into the data cache immediately. (also some further details but not fully understand: Fill data arrives on the data buses. Pending loads sample the data to write into the register file while, in parallel, the caches (instruction or data) also fill using the same bus data. The data cache is write-back, so fills also use its double-pumped capability: The previous cache contents are read out in the same cycle that fill data is written in. The bus interface unit captures this victim data and later writes it back.)

ii Address and control structure

The internal memory system maintains a 32-entry load queue (LDQ) and a 32-entry store queue (STQ) that manage the references while they are in-flight. The LDQ(STQ) positions loads(stores) in the queue in fetch order, although they enter the queue when they issue, out of order.

iii Cache prefetching and management

- Bus interface unit
- Dynamic execution examples

i Store/load memory ordering

ii Load hit/miss prediction