Mar. 2005, Vol.20, No.2, pp.243–249 *J. Comput. Sci. & Technol.*

Microarchitecture of the Godson-2 Processor

Wei-Wu Hu, Fu-Xin Zhang, and Zu-Song Li

Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, P.R. China

E-mail: {hww, fxzhang, lisoon}@ict.ac.cn

Received August 19, 2004; revised January 15, 2005.

Abstract The Godson project is the first attempt to design high performance general-purpose microprocessors in China. This paper introduces the microarchitecture of the Godson-2 processor which is a 64-bit, 4-issue, out-of-order execution RISC processor that implements the 64-bit MIPS-like instruction set. The adoption of the aggressive out-of-order execution techniques (such as register mapping, branch prediction, and dynamic scheduling) and cache techniques (such as non-blocking cache, load speculation, dynamic memory disambiguation) helps the Godson-2 processor to achieve high performance even at not so high frequency. The Godson-2 processor has been physically implemented on a 6-metal 0.18μm CMOS technology based on the automatic placing and routing flow with the help of some crafted library cells and macros. The area of the chip is 6,700 micrometers by 6,200 micrometers and the clock cycle at typical corner is 2.3ns.

Keywords superscalar pipeline, out-of-order execution, branch prediction, register renaming, dynamic scheduling non-blocking cache, load speculation

1 Introduction

The Godson project is the first attempt to design high performance general-purpose microprocessors[1−9] in China. Besides aiming at Linux PCs for office and network servers, they can also be used in many embedded environments such as network computing ends, network switches, industry control, and game machines, etc. Godson implements the MIPS-like instruction set to take advantage of its ready-made hardware systems and software codes.

Multiple levels of parallelism can be explored to improve performance of a processor. In instruction level, out-of-order execution and superscalar technique allow the processor to schedule the execution of instructions in a maximum throughput. In data level, vector instructions that are implemented with SIMD technique enable the processor to generate multiple results with one instruction. In thread level, multithreading enables multiple threads to run simultaneously on a single or multiple processors.

Godson achieves both the architectural and physical goals step-by-step. Architecturally, the Godson-1 processor implements single-issue out-of-order execution pipeline, with static branch prediction and blocking cache; the Godson-2 processor implements superscalar out-of-order execution pipeline, with dynamic branch prediction and non-blocking cache, it also implements some fix-point SIMD instructions by reusing the floating-point datapaths; and the Godson-3 processor will further implement multithreading. Physically, the Godson-1 processor is an ASIC designed chip, Godson-2 will include some customer designed modules, and the later versions of Godson-2 or Godson-3 will be semi- or full-customer designed.

The Godson-2 processor has been physically implemented on a 6-metal 0.18μm CMOS technology based on the automatic placing and routing flow with the help of some crafted library cells and macros. The area of the chip is about 6,700 micrometers by 6,200 micrometers. The clock cycle at typical corner is 2.3ns. Now the first version of Godson-2 chips run well and the Linux PC based on Godson-2 is under development.

The following sections are organized as follows. Section 2 summarizes architectural features of Godson-2. The instruction fetching unit, out-of-order execution engine, functional units, and memory access unit are then introduced in the following four sections. Section 7 briefly presents the physical design and fabrication of the Godson-2 chip. Future work and conclusion are given in Section 8.

2 Godson-2 Processor Architecture Overview

The Godson-2 microprocessor is a general-purpose RISC microprocessor that implements the 64-bit MIPS-like instruction set. It fetches and decodes four instructions per cycle and dynamically issues them to five fully pipelined functional units. Though instructions are executed out-of-order according to their dependency, they are committed in the program order to provide precise exception handling and sequential memory consistency.

The four-way superscalar of Godson-2 raises extremely high requirements for inter-instruction dependency resolving and instruction/data providing. Godson-2 employs out-of-order execution and aggressive cache design to improve pipeline efficiency.

Out-of-order execution is a combination of the register renaming, dynamic scheduling, and branch prediction techniques. Register renaming removes WAR

*Short Paper

The work of this paper is supported by the National High Technology Development 863 Program of China (Grant Nos.2001AA111100 and 2002AA110010) and the Key Knowledge Innovation Project of Chinese Academy of Sciences.

(write after read) and WAW (write after write) dependency and is also essential for precise exception handling and branch misprediction recovering. Godson-2 has a 64-entry physical register file for fix-point and floating-point register mapping respectively. Dynamic scheduling reduces the stall caused by RAW (read after write) data dependency through reordering the instruction. Godson-2 maintains a 16-entry fix-point reservation station and a 16-entry floating-point reservation station for out-of-order instruction issuing. A 32-entry ROQ (reorder queue) ensures that out-of-order executed instructions are committed in the program order. Branch prediction removes control dependency by predicting the direction before the branch instruction is executed. Godson-2 uses a 16-entry BTB (branch target buffer), a 4K-entry BHT (branch history table), a 9-bit GHR (global history register), and a 4-entry RAS (return address stack) for precise branch prediction.

The cache system of Godson-2 also provides potential for high performance. Godson-2 has a 64KB level-one instruction cache and a 64KB level-one data cache, both four-way set associative. It also provides the MIPS R5000 like off-chip cache interface which can support as large as 8MB unified level-two cache. The fully associative TLB of Godson-2 has 64 entries each maps an odd page and an even page. A 16-entry memory access queue that contains a content-addressable memory for dynamic memory disambiguation allows Godson-2 to implement out-of-order memory access, non-blocking cache, load speculation, and store forwarding.

Godson-2 has two fix-point functional units, two floating-point functional units, and one memory access unit. The floating-point units can also execute 32- or 64-bit fix-point instructions and 8- or 16-bit SIMD fix-point instructions through extension of the *fmt* field of the floating-point instructions.

The basic pipeline stages of Godson-2 include instruction fetch, pre-decode, decode, register rename, dispatch, issue, register read, execution, and commit. Fig.1 shows major sections of Godson-2.

In fetch stage, the instruction cache and instruction TLB (Translation Lookahead Buffer) is read according to the content of PC (program counter). Four new instructions are sent to IR (instruction register) if the instruction fetch is TLB hit and cache hit.

In pre-decode stage, branch instructions are found and their branch directions are dynamically predicted.

In decode stage, the four instructions in IR are decoded into internal format of Godson-2 and are sent to the register renaming module.

In register rename stage, a new physical register is allocated for each logical destination register, and the

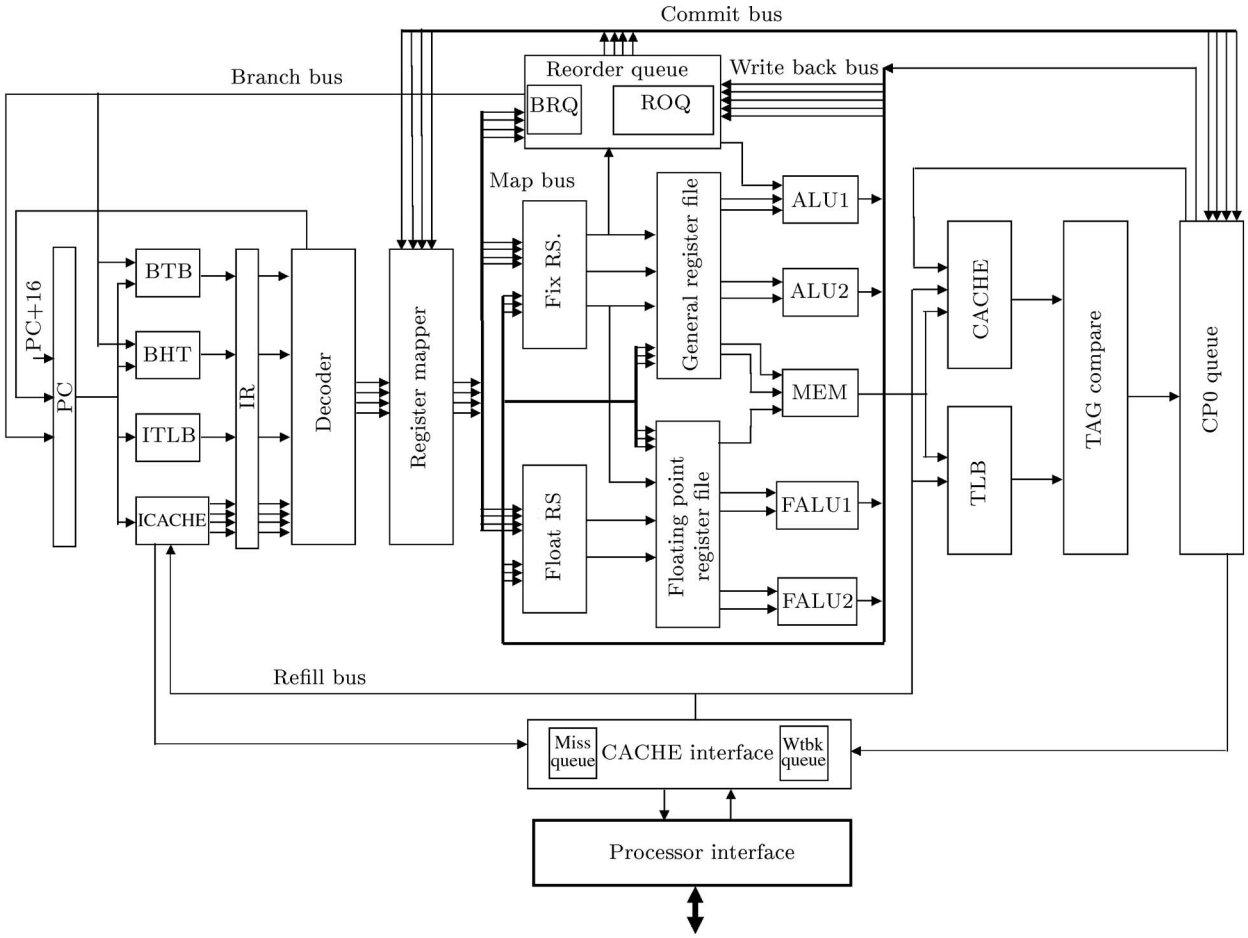

Fig.1. Microarchitecture of Godson-2.

logical source register is renamed to the latest physical register allocated for the same logical register. Inter-instruction dependencies among four instructions mapped in the same cycle are also checked. The renamed instructions are latched for being sent to reservation stations and queues in next cycle.

In dispatch stage, the latched renamed instructions are dispatched to the fix- or floating-point reservation station for being executed, and are sent to the reorder queue for in-order graduation. Associated instructions are also sent to branch queue and memory queue. Each empty entry of reservation stations and queues selects among four dispatched instructions in this cycle.

In issue stage, one instruction with all required operands ready is selected from the fix- or floating-point reservation station for each functional unit. When there are multiple instructions ready for the same functional unit, the oldest one is selected. Each instruction with unready source operands in register mapping stage snoops result and forward buses for their operands.

In register read stage, the issued instruction reads its source operands from the physical register file and is sent to the associated functional units. It may also get the data directly from one of the result buses if its source register number matches the destination register number of the result bus.

In execution stage, the instruction is executed according to its type. The calculation result of the instruction is written back to the register file. Result buses are also sent to the reservation station for snooping and to the register mapping table to notify that the associated physical register is ready.

In commit stage, up to four instructions can be committed in program order per cycle. Committed instructions are sent to the register mapping module to confirm the mapping of its destination register and release the old one. They are also sent to the memory access queue to allow committed store instructions to write cache or memory.

3 Instruction Fetching and Branch Prediction

The Godson-2 pipeline begins with the fetch stage, in which four instructions are fetched in parallel at any word alignment within an eight-word instruction cache line. In each cycle, the processor compares tags read from the cache to physical addresses translated from ITLB (instruction TLB) to select instructions from the correct way. On cache miss a refill request will be raised.

The 16-entry ITLB is a subset of the main TLB. It is different from the main TLB in that each ITLB entry maps only one page. On an ITLB miss, the processor creates an internal Godson-2 instruction which looks for the entry in the main TLB and fills the ITLB. A normal TLB exception will rise if the missing page is not in the main TLB too.

The second stage of the Godson-2 pipeline is the pre-decode stage. In this stage, branch instructions of the four instructions in IR are found and their branch directions and branch targets are predicted. Different branch prediction methods are used for different branch instructions. Branch-like instructions and jump instructions are always predicted taken, allowing the compiler to statically predict branch direction by using branch-like instructions. BHT is used for predicting direction of conditional branch instructions, while BTB and RAS are used for predicting target PC for jump register instructions.

The BHT contains a 9-bit global history register (GHR) and 4K-entry pattern history table (PHT). Each PHT entry has a 2-bit saturating up/down counter. The counter is increased by one if the prediction is correct, and is decreased by one otherwise. The high order bit of the counter is used for branch prediction.

The 16-entry BTB predicts the target PC of the jump register instruction. Each BTB entry contains the PC and target PC of the jump register instruction. Besides, a 2-bit saturating up/down counter is associated with each BTB entry. On replacement, entries with counter values 0 or 1 will be replaced prior to others.

MIPS instruction set does not provide call or return instruction, it normally uses branch/jump and link instruction and the jump register 31 instruction instead. Godson-2 implements a four-entry return address stack. The decoding of a branch and link instruction causes its PC+8 to be pushed to the RAS, while the decoding of a jump register 31 instruction causes the target PC to be popped from the RAS. Each branch instruction saves the top-of-stack pointer of the RAS to repair the top-of-stack pointer of the RAS after branch misprediction.

The third stage of Godson-2 pipeline is the decode stage. In this stage, the four instructions are decoded into internal instruction format of Godson-2 and are sent to the register renaming module. The fix-point multiplication or division instruction is split into two internal instructions because it generates two 64-bit results. Only one branch instruction can be decoded every cycle in this stage to simplify the management of branch instructions.

4 Register Renaming and Dynamic Scheduling

4.1 Register Rename and Register Rename Tables

Register renaming can be implemented either by using separate architectural and rename register files, or by using a merged architectural and rename register file. In the first approach, a stand alone rename register file is used for register rename. In the second approach, rename registers are implemented along with the architectural registers in the same physical register file and a mapping table is used to dynamically map between architectural and physical registers. Godson-2 implements the merged approach and has a 64-entry physical

register file for fix-point and floating-point register rename respectively. Correspondingly, two 64-entry physical register-mapping tables (PRMT) are maintained to build the relationship between physical and architectural registers. Each PRMT entry has the following fields:

• *state*: each physical register is in one of four states, MAP_EMPTY, MAP_MAPPED, MAP_WTBK, and MAP_COMMIT;

• *name*: the identifier of the associated architectural register to which this physical register is allocated;

• *valid*: this bit is used to mark the latest allocation of a given architectural register if more than one physical registers are allocated to it.

The PRMT also includes fields used to restore the register mapping on mispredicted branch canceling.

In register rename stage, the PRMT is associatively looked up for the two source registers *src*1, *src*2 and the destination register *dest* of each instruction to find the associated latest mapped physical register *psrc*1, *psrc*2, and *odest*. Besides, a free physical register *pdest* whose state is MAP_EMPTY is allocated to the destination register *dest*, and the state of the newly allocated physical register is set to MAP_MAPPED. The valid bit of the *pdest* entry is set to "1" and the valid bit of the *odest* entry is set to "0" reflecting that *pdest* becomes the latest allocated physical register for the *dest* architectural register.

Since four instructions are mapped concurrently, inter-instruction dependencies among instructions mapped at the same cycle should be checked. If the source register *src*1 of an instruction is identical to the destination register *dest* of a previous instruction mapped at the same cycle, the physical register corresponds to *src*1 should be *pdest* of this previous instruction, rather than the *psrc*1 looked up from the PRMT. This is also true for *psrc*2 and *odest*.

After register renaming, the architectural register name *src*1, *src*2, and *dest* are replaced with the physical register name *psrc*1, *psrc*2, and *pdest*. These physical register names are sent to the reservation station. The *odest* field is kept in the reorder queue and is used to release the physical register when the instruction is committed. The processor determines dependencies simply by comparing the physical register names.

After an instruction is executed, its associated PRMT entry is set to MAP_WTBK such that the following instructions reading this physical register know the value is ready.

When an instruction is committed, it sets the *pdest* entry of PRMT to MAP_COMMIT and the *odest* entry to MAP_EMPTY, that means its destination register value is regarded as the processor state and the previous value for this destination register is discarded.

It can be seen from the above register rename process that there may be multiple physical registers allocated to the same architectural register because a logical register may have a sequence of values as it is written by instructions in the pipeline. Physical registers assigned to the same logical register hold both committed values and temporary results as instructions flow through the pipeline. A physical register is written exactly once for each assignment of it.

4.2 Instruction Issue and Reservation Stations

Register renamed instructions are latched and then sent to the reservation station to be scheduled for execution. Godson-2 has two independent group reservation stations. Fix-point and memory instructions are sent to the fix-point reservation station. Floating-point instructions are sent to the floating-point reservation station. Each reservation station has 16 entries and can accept as many as four instructions per cycle.

In the register rename stage, the PRMT is looked up to see whether the associated operand has been generated and written back to the register file. Result buses and forward buses are also snooped for renaming and dispatching instructions and for instructions in reservation stations to decide when the required operand will be ready in the register file. The associated ready bit is set to ready if the destination register of one snooped bus matches the source register of the snooping instruction. Result and forward buses stem from the five functional units. The result buses send out the execution results of functional units, while the forward buses forecast which result will be sent out in next cycle.

The reservation stations can issue as many as five operand-ready instructions to the five functional units. If there are multiple operand-ready instructions for the same functional unit, the oldest one is issued. To record the age of each instruction, an *age* field is added to each entry of the reservation station. It is set to a low value when an instruction enters the reservation station, and is increased by one each time an instruction of the same functional unit enters the reservation station.

Unlike MIPS R10000 which has a separate in-order reservation station for memory accesses, Godson-2 issues memory accesses through the fix-point reservation station. To keep the order of some special CP0 (coprocessor 0) instructions, such as control register instructions, TLB instructions, and cache instructions, Godson-2 imposes two types of order relations in the fix-point reservation station. The first type is called "wait issue" relation, "wait issue" instructions cannot be issued until all its previous instructions have been committed. The second type is called "stall issue" relation. "stall issue" instructions stall the issue of following instructions until it is committed. Ordinary memory access instructions are neither "wait issue" nor "stall issue".

4.3 Reading Operands and Register Files

Godson-2 has one fix-point physical register file and one floating-point physical register file, both with the size of 64×64. Issued instructions read operands from

the register file before they are sent to functional units for execution.

The fix-point register file has three write ports and six read ports. The two fix-point units and the memory unit use one write port and two read ports each. The floating-point register file has three write ports and five read ports. The two floating-point units use one write and two read ports each. Besides, floating-point load instructions use one write port and floating-point store instructions use one read port of the floating-point register file. Move instructions between fix-point and floating-point register files such as MTC1, DMTC1, MFC1, DMFC1, CTC1, and CFC1 use the memory access data path to transfer data and hence are executed by the memory unit.

Special operands such as the program counter of branch and link instructions or the predicted *taken* bit of conditional branch instructions are read from the branch queue and sent to the associated functional unit in parallel with operands from register file.

4.4 Instruction Commit and Reorder Queue

The reorder queue holds all instructions after register mapping and before they are committed. After instructions are executed and written back, the reorder queue commits them in the program order. The reorder queue can hold as many as 32 instructions concurrently.

Reorder queue can accept as many as four mapped instructions per cycle. Newly entered instructions are set to ROQ_MAPPED state. After the instruction is written back, its state in reorder queue is set to ROQ_WTBK for ordinary instructions and ROQ_BRWTBK for branch instructions. The state of branch instructions are set to ROQ_WTBK after the branch result has been sent to other parts of the processor through the branch bus to justify branch prediction tables and to cancel instructions following mispredicted branches. ROQ_WTBK instructions can be committed if they reach the head of the reorder queue.

Reorder queue graduates instructions in order. It commits as many as four ROQ_WTBK instructions in the queue head per cycle. The execution result of an instruction cannot be made as the processor state before this instruction graduates. When an instruction graduates, its *pdest* and *odest* fields are sent to the register mapping module to confirm the mapping of *pdest* entry as the processor state and to free the mapping of *odest* entry, it also informs the memory queue that the corresponding store instructions can start to modify the memory.

For precise exception handling, exceptions are not processed as soon as they occur. They are recorded in the reorder queue instead. When the exception instruction reaches the head of the reorder queue, the exception information is sent out through exception bus. All the following instructions are cancelled, exception information is recorded in CP0 registers, and the PC is set to the entry point of exception handler.

4.5 Branch Canceling and Branch Queue

A branch instruction enters the branch queue at the same time when it is sent to the reorder queue and the reservation station. At most, one branch instruction can be accepted by the branch queue per cycle. The branch queue can hold as many as eight branch instructions concurrently.

The branch queue provides the information necessary for execution when a branch instruction is issued to be executed. The information includes the PC value for branch and link instructions, and the predicted *taken* bit for conditional branch instructions.

After a branch instruction is executed, execution results specific to branch instructions are written back to the branch queue. The results include the target PC for JR and JALR instructions, the branch direction for conditional branch instructions, and a bit indicating whether the branch prediction is error. The branch instruction execution result should be feedback to the instruction fetch part before it can be committed. Besides canceling mispredicted instructions, the branch execution result is also used to justify the BHT, BTB, RAS, and GHR for following branch prediction.

For incorrect prediction, instructions that follow the mispredicted branch should be cancelled. The key issue is for each instruction in the pipeline to decide whether it is before or after the mispredicted branch. Godson-2 divides the continuous instruction stream into basic blocks separated by branch instructions. Each instruction is assigned a branch queue position identifier *brqid* that can be regarded as its basic block number. For branch instruction, this identifier indicates its position in the branch queue; for ordinary instruction, this identifier indicates its previous branch instruction position in the branch queue. In this way, each instruction can determine its relative position to the mispredicted branch by comparing its *brqid* with the *brqid* of the mispredicted branch. For example, if the head of the branch queue is in position 0 and the mispredicted branch is in position 5, then an instruction with the *brqid* value of 6 will be cancelled and an instruction with the *brqid* value of 4 will not be cancelled. Delay slot instructions should be paid special attention in branch canceling.

5 Functional Units

Godson-2 has two fix-point functional units, ALU1 and ALU2, and two floating-point functional units, FALU1 and FALU2.

ALU1 executes fix-point addition, subtraction, logical, shift, comparison, trap, and branch instructions. All ALU1 instructions are executed and written back in one cycle.

ALU2 executes fix-point addition, subtraction, logical, shift, comparison, multiplication, and division instructions. Fix-point multiplication is fully pipelined and has a latency of four cycles. Fix-point division uses the SRT algorithm and is not fully pipelined, the latency of fix-point division ranges from 4 to 37 cycles depending on the operands. All other ALU2 instructions can be executed and written back in one cycle.

The fully pipelined FALU1 executes floating-point addition, subtraction, absolute, negation, conversion, comparison, and branch instructions. The floating-point absolute, negation, comparison and branch are two-cycle instructions, while the latency of floating-point addition, subtraction, and conversion instructions is four-cycle.

FALU2 executes floating-point multiplication, division, and square root instructions. The fully pipelined floating-point multiplication uses two-bit Booth-encoded Wallace tree algorithm and has a latency of five cycles. The division and square root use the SRT algorithm and are not fully pipelined. The latency of single/double precision floating-point division ranges from 4 to 10/17 cycles, the latency of single/double precision floating-point square root ranges from 4 to 16/31 cycles, depending on the operands.

Besides executing all MIPS III floating-point instructions, the floating-point functional units can also execute paired-single floating-point instructions which calculate two single precision operations (addition, subtraction and multiplication) in the 64-bit datapath, 32- or 64-bit fix-point instructions (arithmetic, logic, shift, compare, and branch), and 8- or 16-bit SIMD fix-point instruction through extension of the *fmt* field of the floating-point instructions.

6 Memory Access and Memory Management

The memory subsystem of Godson-2 also provides potential for high performance. Godson-2 has a 64KB level one instruction cache and a 64KB level one data cache, both four-way set associative. The fully associative TLB of Godson-2 has 64 entries each maps an odd and an even page. A 16-entry memory access queue that contains a content-addressable memory for dynamic memory disambiguation allows Godson-2 to implement out-of-order memory access, non-blocking cache, load speculation, and store forwarding.

The Godson-2 memory access pipeline is split into four stages. Memory references are issued out-of-order to the address calculation unit. Calculated virtual addresses and values are sent to TLB and cache unit; TLB translates virtual addresses into physical addresses in parallel with cache accessing; TLB and cache results are then used to determine whether cache access is hit and are sent to memory access queue, where dynamic memory disambiguation and memory forwarding is performed; finally the results are written back when ready.

The interface of the Godson-2 processor supports split read and R5000 like external level two cache. The size of the external cache ranges from 256KB to 8MB.

6.1 TLB

Godson-2 implements a 64-entry fully associative TLB for virtual-physical address translations. It contains a CAM part that is used to associatively search virtual addresses and an RAM part which stores physical page numbers and page protect bits. The CAM lookup is done in address calculation stage to avoid the need of asynchronous RAM. To reduce hardware cost, Godson-2 uses 40-bit virtual address and 36-bit physical address instead of the rarely needed 64-bit.

One important feature of Godson-2 is its ability to do page level execution protection. It is implemented as an additional bit in TLB entries, which can be manipulated by software. An address error exception will occur in an instruction fetch attempt from a page with the bit set. Operating systems can utilize this ability to effectively invalidate many exploits of buffer overflow vulnerabilities, which are the most common form of security holes.

Many of the MIPS coprocessor 0 control registers are related with TLB, so they are implemented close-by to minimize wiring. Some control register bits affect instruction fetch or other processor status hence CP0 instructions which access control registers are serialized in instruction fetch or issue stage.

6.2 Data Cache

Godson-2 comes with a 64KB four-way set associative primary data cache. It is virtually indexed and physically tagged so that accesses can happen in parallel with TLB lookups. The replacement policy is random, but two continuous replacement of the same block is avoided by hardware.

To reduce chip area and ease physical design, single port RAM is used for both tag and data. Godson-2 allows simultaneous loads and write-back of stores provided they access different banks to alleviate cache access conflict. When cache port conflict does occur among refills, loads (stores read only the tag array) and write-back of stores (which write cache data only), refills have the highest priority while write-back of stores have the lowest priority.

Because each cache way contains 16K bytes (four times the minimum virtual page size), two of the virtual index bits (13:12) might not equal the bits in the physical address tag. Current Godson-2 does not provide hardware to prevent possible consistency problems. Operating systems are expected to provide solutions here. There are already mature software solutions available to solve inconsistent problems caused by virtual index.

6.3 Memory Access Queue

Memory access queue is the core unit of Godson-2 memory subsystem. It can track up to 16 in-flight

memory loads or stores. Loads and stores enter the queue out-of-order, but an in-order architectural memory model is maintained. Multiple cache misses and hit under miss are allowed.

Godson-2 does not retry a memory access in case of cache miss or hazards. Using a physical address CAM, the memory access queue dynamically performs disambiguation and forwarding between accesses. When a load enters the queue, it checks all the older stores for possible bypass for each byte it needs. When a store enters the queue, it checks all the younger loads in front of another tyounger store to the same byte in the queue to decide whether to forward value to them. The queue also snoops cache refill and replace operations.

The queue has four read ports. The first read port is used to select first result-ready instruction and write back its result. Cache hit loads are written back even when there is pending stores before it. If late coming store should forward its value to the speculatively written back load, the load and its following instructions will be cancelled. The second read port is used to select the first committed write-ready store and write its value to data cache. A store is write-ready when the value to store is valid and it has been committed (that is, cannot be cancelled). The third read port is used to issue miss request to the next level memory. Uncached accesses and exception handling use the last read port.

7 Physical Implementation

The above design of the Godson-2 processor has been physically implemented based on the ASIC flow with some manual placement and a number of crafted cells and macros. Design Compiler was used to do logical synthesis, Physical Compiler was used to generate the placement for cells, and Astro was used for floorplan, clock tree generation, and routing. The design was based on SMIC's (Semiconductor Manufacturing International Corporation) 6-metal 0.18μm CMOS process.

A number of crafted cells and macros were built for this design. The crafted cells include some basic cells such as flip-flops with much lower latency and power than those of Artisan's library, NANDs, NORs, AOIs, MUXs, buffers and inverters with different sizes; some double height cells such as 4-, 6-, or 8-bit comparator, 4-bit flip-flops, and full adder; and the pad cells for flip-chip package. The crafted macros include a 64×64 register file with 3 write ports and 6 read ports and a special RAM macro for TLB.

To reduce clock cycle time, some datapath modules or modules with replicated structure were manually mapped to the cell library from a structural Verilog model, and were manually placed in a bit-sliced way. These modules were automatically routed in a flatten way after the detailed placement. Besides, the useful clock skew technique is used for critical path pipeline stage to borrow time from adjacent pipeline stages.

The area of the chip is about 6,700 micrometers by 6,200 micrometers. The clock cycle at typical corner is 2.3ns. The power consumption is about 2.0–3.0 watt in 400MHz. Now Godson-2 chips are running well and a type of Linux PC based on the Godson-2 processor is under development.

8 Conclusion and Future Work

This paper introduces the micro-architecture of the Godson-2 processor. Godson-2 is a 64-bit, 4-issue, out-of-order execution RISC processor which implements the MIPS-like instruction set. The adoption of the aggressive out-of-order execution techniques (such as register mapping, branch prediction, and dynamic scheduling) and cache techniques (such as non-blocking cache, load speculation, dynamic memory disambiguation) helps Godson-2 to achieve high performance even at not so high frequency.

Our recent work includes further improving the functional units, the instruction fetch system and memory system. Improvements on the functional units include the support of conditional move instruction and the floating-point multiply-add instruction. Improvements on instruction fetch system include more precise branch prediction and support for SMT (Simultaneous Multithreading). Improvements on memory system include on-chip level-two cache implementation, cache coherence for SMP and on-chip DDR memory interface. Besides, custom physical design of the next version of Godson-2 processor with more advanced technology is being undertaken.

Our future work includes implementing a special Java co-processor and exploiting multithreading parallelism through putting multiple processors in the same chip.

References

[1] Weiwu Hu, Zhimin Tang. Microarchitecture design of the Godson-1 processor. *Chinese Journal of Computers*, April 2003, pp.385–396. (in Chinese)

[2] Divid Patterson, John Hennessy. Computer Architecture: A Quantitative Approach. Morgan Kaufmann Publishers, Inc. 1996.

[3] Kessler R. The Alpha 21264 microprocessor. *IEEE Micro*, March/April 1999, 19: 24–36.

[4] Kenneth Yeager. The MIPS R10000 superscalar microprocessor. *IEEE Micro*, April 1996, 16: 28–41.

[5] Tim Horel, Gary Lauterbach. UltraSparc-III: Designing third-generation 64-bit performance. *IEEE Micro*, May/June 1999, 19: 73–85.

[6] Ashok Kumar. The HP PA-8000 RISC CPU. *IEEE Micro*, Mar.–Apr., 1997, 17: 27–32.

[7] Joel Tendler, Steve Dodson, Steve Fields *et al.* Power 4 system microarchitecture. *IBM Technical White Paper*, October 2001.

[8] Huck J *et al.* Introducing the IA-64 architecture. *IEEE Micro*, Sept.–Oct., 2000, 20: 12–23.

[9] Glenn Hinton, Dave Sager, Mike Upton *et al.* The microarchitecture of the Pentium 4 processor. *Intel Technology Journal*, Q1, 2001.