A superscalar AMULET

V. A. Chouliaras (vc@cs.man.ac.uk), D. A. Edwards (doug@cs.man.ac.uk)
AMULET Group, Department of Computer Science
The University, Oxford Road, Manchester M13 9PL
UK

Abstract: A 4-way superscalar model for the ARM instruction set is proposed. The architecture utilises register renaming and dynamic instruction scheduling to achieve a high instruction execution rate. A reorder buffer scheme provides a mechanism for result-forwarding and in-order processor state updates. Performance-inhibiting features of the ARM instruction set like conditional mode-update instructions and user traps (SWIs) are dealt through a novel locking mechanism in the register mapping table.

Introduction

As part of it's continuous investigation into asynchronous implementations of the ARM architecture, the AMULET group of the University of Manchester is finalising the architectural features of the third incarnation of the asynchronous ARM processor.

AMULET 1 [Pave94] and AMULET 2 have been successful designs proving the feasibility of applying an asynchronous design framework to a highly complicated problem as is the implementation of a well-established RISC architecture.

The previous two attempts focused on power and area-efficient implementations through utilising relatively simple scalar decode/execute pipelines with strict, in-order execution; AMULET 3 however shall be a performance-directed architecture which though utilising a scalar pipeline, shall permit for out-of-order execution of memory and ALU operations [Gilb96].

In parallel to AMULET 3 activities, a superscalar architecture for the ARM processor is currently being investigated. A number of features of the ARM instruction set seem to be performance-limiting in the context of multiple, free-flowing pipelines with instructions at varying stages of completion. These features are the conditional instructions and conditional mode-update instructions in particular and software interrupts. These instructions can potentially cause a significant book-keeping overhead in order to present a consistent processor mode to subsequent instructions due to the 'distance' between instruction decode and instruction graduation. Furthermore, the small number of ARM registers compared to other RISC architectures, is expected to limit the potential ILP that can be gained in a purely score-boarded architecture.

To circumvent these issues, register renaming is utilised to provide a distinct storage location for the results of each instruction being dispatched. Coupled with a mechanism for the dynamic disambiguation of the processor mode due to conditional mode-update instructions (possibly in an abort/branch shadow), and a result-forwarding mechanism, it is expected that the proposed architectural model shall allow deep out-of-order issue and recovery of a precise state.

Proposed Architecture

A high-level block diagram of the proposed architecture is depicted in Figure 1.

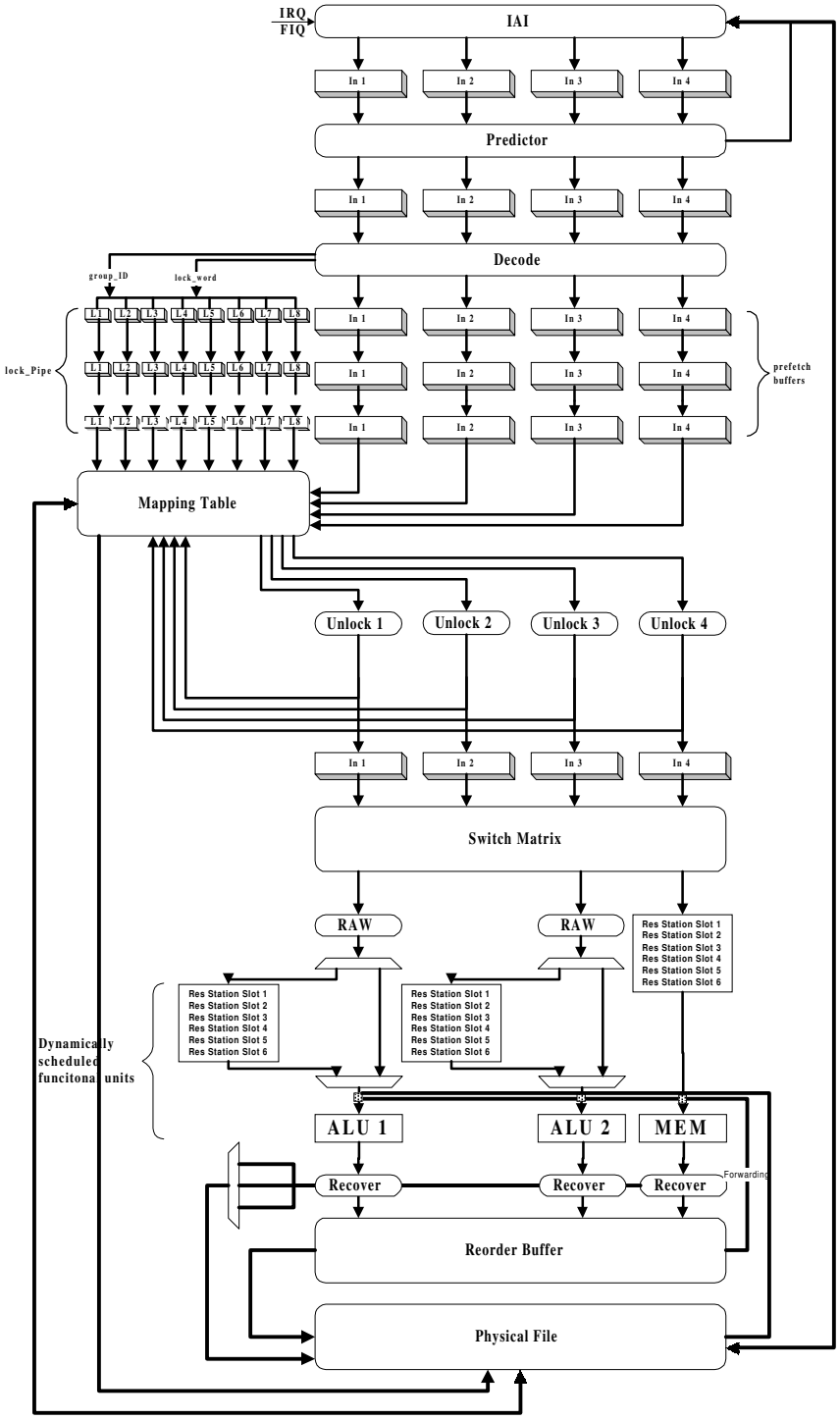

Figure 1: High level block diagram

Instruction Address Interface

The instruction address interface (IAI) can be considered as part of a decoded instruction cache: a cache line (4X32 bits) is brought in and any non-aligned instructions are tagged as invalid. Invalid instructions do not participate further in branch prediction, instruction decode, register renaming and dispatch to functional units. This scheme avoids the complexity of having to fetch from two cache lines at the expense of sacrificing between 1 and 3 instruction decode/execute slots for non-aligned program counters.

Branch predictor

In a dynamically scheduled asynchronous microprocessor, the time of instruction graduation[1] is bounded but non-deterministic. A bypass mechanism is therefore essential to maintain an instruction window of instructions awaiting for operand availability in order to commence execution. Control-flow changes can deplete that instruction window while the target address computation is being scheduled for dynamic execution. The problem is more severe in superscalar processors where there are multiple instances of the same pipeline stages sitting idle [Johnson91]. A consequence of this statement is that a sophisticated branch-prediction mechanism is required to fully utilise the decode/execute bandwidth of the multiple pipelines.

The proposed scheme is based on a single-level of associative memory maintaining two bits of history for each branch in the instruction cache, as utilised on the Intel Pentium processor. More sophisticated mechanisms like a correlating predictor [Henn96] and the scheme proposed in [Roten96] are being investigated.

Decode

An original design goal was the requirement to keep the synchronization point as close to the instruction cache as possible. This has been achieved through the use of a locking mechanism that ensures:

1. Instructions beyond decode carry with them sufficient information to do an ordered or independent register renaming operation.
2. If an instruction decode pipeline is blocked due to some long-latency operation, the remaining pipelines can still decode and dispatch instructions, as long as the later instructions do not require the physical address of the registers being accessed by the blocking instruction. This is ensured by a re-circulating token-passing lock mechanism originating at instruction decode.

Decode is thus the last point of synchronization in the current model. A simpler architectural model would synchronise instruction lines after decode, possibly eliminating some or all of the prefetch buffers and the token lock mechanism. There are no figures as yet available to support the preference of that scheme other than the reduced hardware complexity it implies

A number of processes are forked at decode:

1. Mode-altering instructions are identified: a conditional mode-update instruction shall lock out logical registers R_8 through R_{14} in the mapping table until it successfully graduates. At that point, a definite mode update of the logical registers R_8 through R_{14} mapping can occur.
2. Register dependencies across the cache line are identified: the source and destination registers of all four instructions are identified in parallel. Dependent instructions (figure 2) are tagged with special control information to be passed to the mapping table rename controllers in order to ensure correct sequencing of the renaming operation via a conditional interlock mechanism [vac96].
3. A single *group lock word* is built and pipelined down the lock pipe: each instruction group originating from the decode is being assigned a unique tag. The same tag is pipelined down the lock pipe to ensure that if, for any reason the original group is partly blocked prior to reaching decode, no future instructions can access the source and destination registers of the original group until the latter have been read/renamed respectively.
4. Unlock words, one per instruction in the line are built and attached to each instruction: for non-mode altering instructions the logical registers are unlocked directly after rename. Conditional

[1] Instruction successfully updating the processor state after the previous instruction has done so.

mode-changes however will not unlock R_8 through R_{14} until they successfully conclude and are not in a branch/abort shadow.

5. Assigns a Group_ID to each instruction line: this is the token-assignment step of the locking mechanism.
6. Ordered-renaming control information is attached to each instruction of a dependent instruction group.
7. Reorder buffer locations are assigned to group instructions: each instruction in the group is assigned the next available reorder buffer entry.
8. Memory order buffer entries are allocated to memory instructions: memory accesses go through a modified FIFO referred to (rather arbitrarily) as the memory order buffer. The characteristic of this structure is that memory accesses must be both at the end of the queue and have their source operands available in order to compute a virtual address and be dispatched to the MMU. Furthermore, an aborting load/store shall not permit further memory accesses.
9. Splits LDMs & STMs: multiple load/store operations will be dispatched as single LDRs/STRs [Gilb96b] with the same PC to cater for an aborting situation. If that is not the case, the decode paths need to be made extremely wide in order to inform the rename unit that an LDM (possibly more) with up to 15 destinations needs to rename all it's destination registers.
10. Reservation-station slots are assigned to each instruction: due to pipeline length, it is unknown until the final stage prior to execution, whether there are pending updates of the source registers of an instruction. As a result, the read-after-write (RAW) check point is the last point of decision on whether an instruction can execute immediately or needs to be bypassed to a reservation station. Pre-allocation of reservation station entries during instruction decode ensures that the time-critical RAW-check is not a bottleneck.

Mapping Table

The decoded instruction lines with register renaming control information attached exit the decoder and proceed fully asynchronously to rename their destination registers.

All general-purpose destination registers are renamed, including the CPSR, to a distinct physical location residing in the physical register file. The current, logical to physical relation is held in the logical file (mapping table), along with the current processor mode, as supplied by the reorder buffer. If a conditional mode update was dispatched at an earlier time and hasn't completed yet then processor mode is currently ambiguous; the conditional mode-change instruction 'locks-out' instructions from renaming registers R_8 through R_{14}[2] until a consistent processor mode is recovered.

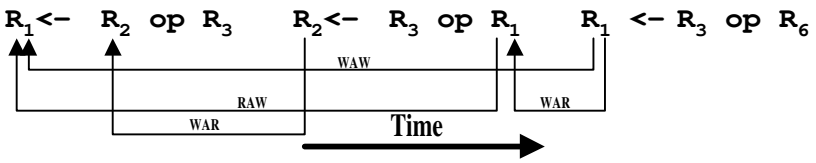

Figure 2: Dependent group of 3 instructions

Switch Matrix

The instruction packets originating from the renaming unit are ready for execution, subject to availability of their operands. A switch matrix arrangement in the from depicted in Figure 3 is utilised to direct each instruction packet to the appropriate dynamic execution path.

[2] Though it permits all instructions having a destination register other than R8..R14 to proceed.

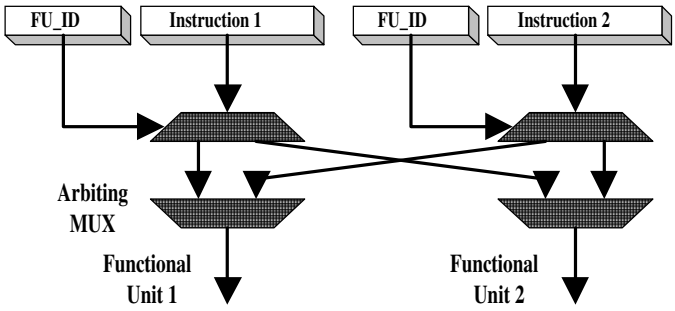

Figure 3: 2X2 switch matrix

RAW-check

A unary-encoded, physical-register availability field is maintained in the core to indicate validity of data either in the reorder buffer or the physical register file. An instruction renaming it's destination registers marks these registers as unavailable; future instructions requiring the contents of these registers shall retire to the reservation stations and continuously monitor the register availability flag.

The field is updated by:

1. Instructions entering the reorder buffer: successful instructions will write their entry in the reorder buffer in parallel to indicating the availability of their destination register result.
2. Instructions that failed their condition codes: a special recovery stage is required by instructions that failed their condition codes. This is covered in a later section.
3. The reorder buffer: the reorder buffer writeout process shall update the register availability flag and IAI colour if an abort was identified.

Reservation stations

A dynamic scheduling scheme is utilised to hold instruction packets until all RAW hazards are resolved and operands are available either from the reorder buffer or from the physical register file.

The current model defines two such 'instruction pool' architectures: for non-memory and memory functional units.

Non-memory functional units

The general from of the reservation station is depicted in Figure 4. Instruction packets not ready to execute enter a location assign to them during decode. Prior to doing so, they build a unary patter of all their source registers.

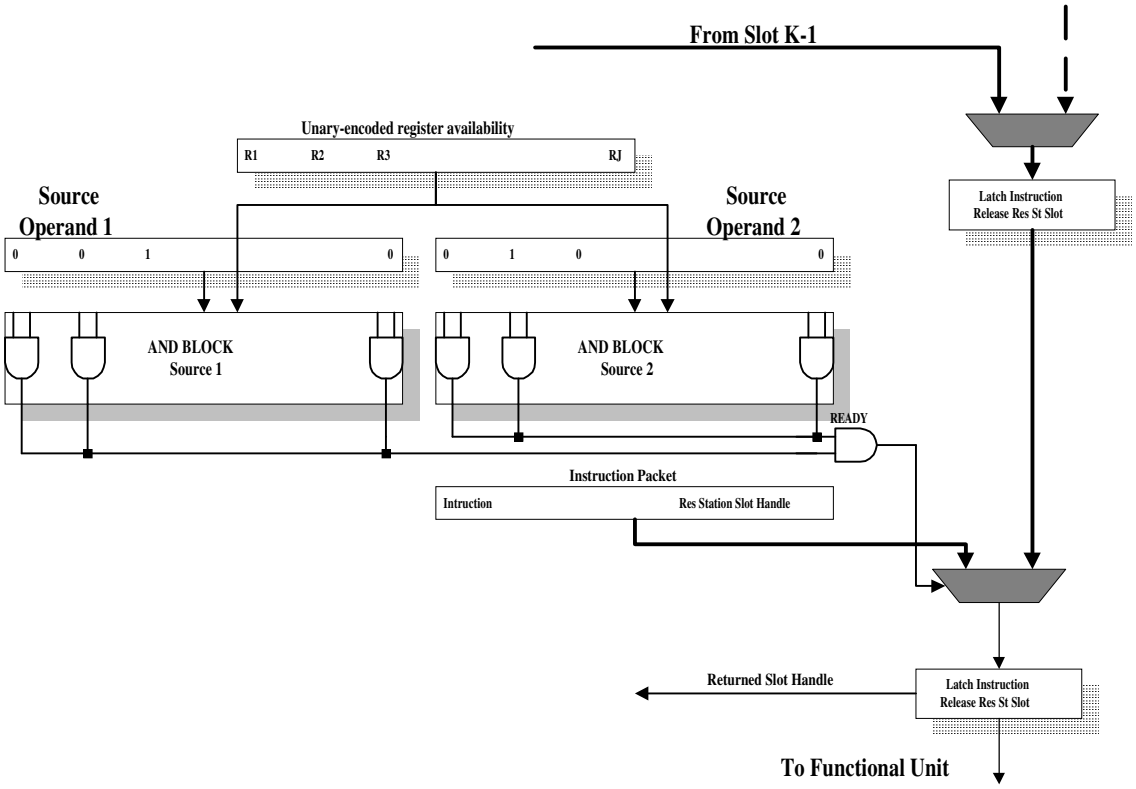

Figure 4: Non-memory functional unit reservation station

Memory system

Though at an early point, the current model assumes all memory references taking place in-order with respect to each other, out-of-order with respect to operations in non-memory functional units. More sophisticated schemes will be investigated in order to decide upon an effective memory-disambiguation mechanism.

Recover point

Instructions that have failed their condition-code check, bypass the functional unit they should execute and enter a special recovery unit in order to re-establish the correct logical-physical relation by doing an internal register transfer of the old contents of their destination registers to the new, renamed address of these registers.

The transfer is essential due to the mapping table being part of the processor state and being updated always, irrespective if an instruction will fail it's condition codes or is in an abort/branch shadow.

Reorder Buffer

The reorder buffer mechanism implements two basic functions:

1. Ensures in program-order processor-state updates.
2. Forwards results to instructions in the execution paths.

The reorder buffer mechanism shall be implemented in the scalar AMULET 3 processor [Gilb96].

Physical Register File

The physical file maintains a pool of RAM locations assigned to specific logical registers during register renaming and associated book-keeping information to ensure prompt return of invalid RAM locations to the global register pool.

Current Status of the model

This report summarises a more or less finalised superscalar architectural model for the ARM instruction set. The most essential architectural features, that is, the early synchronisation point, the conditionally interlock pipelines, dynamic instruction scheduling and processor mode update mechanism are expected to go through additional evolution stages prior to committing to a high-level description.

Processor modelling shall take place in either a standard high-level language like C or in one of the special HDLs currently in development by members of the AMULET Group [Bards96], [Ende96].

References

[Bard96] A. Bardsley, *'An asynchronous Logic Synthesiser'*, internal report, Department of Computer Science, University of Manchester.

[Ende96] P. B. Endecott, *'A Hardware Description Language for Asynchronous Design'*, private communication, submitted to Async97, Department of Computer Science, University of Manchester.

[Gars94] J. D. Garside, *'Whither AMULET2?'*, AMULET Modelling Workshop, Windermere, Cumbria, July 1994.

[Gilb96] D. A. Gilbert, J. D. Garside, *'A Register Forwarding Mechanism for Asynchronous Pipelined Systems'*, private communication, submitted to Async97, Department of Computer Science, University of Manchester.

[Gilb96b] D. A. Gilbert, *'VHDL Modelling of the AMULET3 processor'*, Ph.D. Research, Department of Computer Science, University of Manchester. Publication expected March 1997.

[Henn96] Hennesy & Patterson, *'Computer Architecture : A Quantitative Approach'*, Second Edition, 1996, Morgan Kaufmann.

[Johnson91] M. Johnson, 'Superscalar Microprocessor Design', Prentice Hall, 1991.

[Pave94] N. C. Paver, *'The Design and implementation of an asynchronous Microprocessor'*, Ph.D. Thesis, Department of Computer Science, University of Manchester, WWW accessible: http://www.cs.man.ac.uk/amulet/publications/thesis/paver94_phd.html.

[Roten96] E. Rotenberg, S. Benett, J. Smith, *'Trace Cache: A Low-Latency Approach to High Bandwidth Instruction Fetching'*, Multiscalar Project, University of Wisconsin-Maddison, WWW-accessible.

[vac96] V. A. Chouliaras, D. A. Edwards, *'DDANIA: The Don't Dare Not Issue Architecture'*, submitted to Async97, Department of Computer Science, University of Manchester.