.

The 16-Fold Way: A Microparallel Taxonomy

Barton J. Sano and Alvin M. Despain
(sano@acal.usc.edu, despain@acal.usc.edu)

Advanced Computer Architecture Laboratory
University of Southern California Los Angeles

Abstract

This paper presents a novel microparallel taxonomy for machines with multiple-instruction processing capabilities including VLIW, superscalar, and decoupled machines. The taxonomy is based upon the static or dynamic behavior of four abstract, operational stages that an instruction passes through. These stages are fetch, decode, execute, and retire. This two valued, four variable taxonomy results in sixteen ways that a processor's microarchitecture can he specified. This paper categorizes different machine instances that are either actual implementations or proposed systems within the taxonomy framework. Four new processor microarchitectures are postulated which provide additional features and are instances of the remaining unexplored microparallel classifications.

Keywords: computer taxonomy, microparallel machines, static and dynamic behavior.

1.0 Introduction

At the gross organizational level the language of computer architecture abounds with the usage of Flynn's structural taxonomy of four classifications; SISD, SIMD, MISD, and MIMD [Flynn72]. His taxonomy characterizes how instruction and data streams interrelate as well as how processors are organized to achieve various levels of parallelism. For example, his term SIMD refers to an organization "Single-Instruction stream Multiple-Data stream" in which a computer executes one instruction stream but each instruction has a set of data items to which the operation is applied. A classic example of this organization is the CRAY-1 with its vector operations. The primary purpose of this paper is to describe the various ways instruction-level parallelism is achieved at the next lower level of implementation, the microarchitecture level underlying the classifications of Flynn's taxonomy.

Although superpiplined and superscalar machines similarly exploit instruction-level parallelism [Jouppi89], this paper concentrates on processors capable of multiple-instruction issue or execution per cycle identified as *microparallel* machines. This general machine category includes VLIW, superscalar, as well as decoupled machines (Figure 1).

Here a VLIW architecture provides parallel, tightly-coupled functional or memory units and completely relies upon the compiler to explicitly schedule instructions or components of the long word for execution. A superscalar organization differs from a VLIW in that it provides more sophisticated dynamic instruction scheduling capability, either in the form of out-of-order issue or execution. The third machine type is a decoupled organization which typically has multiple instruction streams each allowed to flow at the rates determined by the separated functional units.

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

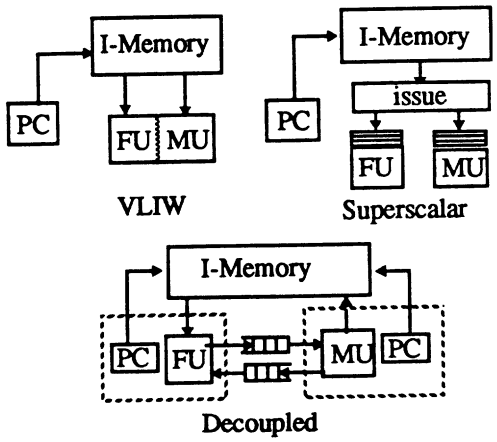

Figure 1. Microarchitectures under consideration.

Recently, there have been several proposals for classifying processor microarchitectures and analysis frameworks for primarily superscalar machines [Chang91, Fisher91, Kuga91, Murakami89]. None satisfactorily classify all the above microparallel machines types. Additionally, these classification schemes are primarily based on specific implementations which can limit the design space to existing microarchitectures instead of pointing to new alternative organizations. As a result, we introduce the *16-Fold Way*, a taxonomy based first on the abstract behavior of the processing steps rather than a particular structure or implementation.

This paper is organized as follows: Section 2 describes this microparallel taxonomy along with its various component static and dynamic behavior. Section 3 categorizes established machine instances out of the sixteen possible taxons in terms of the static and dynamic behaviors. Section 4 then describes four new microarchitectures from the remaining unexplored categories and demonstrates how the taxonomy can be used to postulate alternative designs. Finally, conclusions are made in Section 5.

2.0 Microparallel Taxonomy

All three of the above microparallel organizations (Figure 1) can be described by the behavior of a pipeline of four operational stages at the microarchitecture level which consists of a fetch (F), decode (D), execute (E), and retire (R) stage (Figure 2). The first stage fetches from memory instructions placed in order by the compiler and produces a stream of instructions. The next stage then decodes these instructions, gathers operands or possibly symbolic values for the operands, and dispatches the instructs to the execute stage. This execution stage is a collection of parallel and possibly pipelined functional units and produces arithmetic as well as memory access results. These results are finally committed in the retire stage which finally forms the state of the processor. Although the diagram below shows a single stream of instructions, we can have multiple instructions in a single stream entering and ending a stage (see Section 2.1).

This simple micropipelined processor model conceptually processes the stream of instructions at each stage in the order as they were initially fetched from memory. In contrast, a microparallel execution processor might transform or permute the instruction stream at any stage. This distinction between in-order and out-of-order processing is a useful behavioral abstraction that can describe all four operational stages and will be denoted as *static* and *dynamic* respectively. Thus, any stage that can change the number or order of the instructions with respect to the input of the stage will be considered dynamic. These two behavioral values (static vs. dynamic) and four variables (FDER) provide sixteen different ways to describe the behavior as well as the unique capabilities of a particular machine organization.

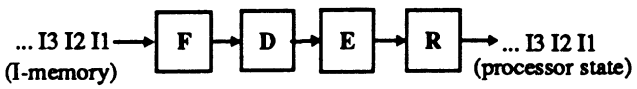

Figure 2. Processor Stages.

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

The taxonomy is organized in a "K-map" (Figure 3) for ease of comparison and positions the fully static processor classification (i.e. SSSS) in the upper left hand corner. This processor class is influenced entirely by the program order specified by the compiler, while categories with a dynamic stage represent machines with some capability to dynamically alter the processing order of the instructions. For instance, the machine classifications in the bottom two rows all have a dynamic fetch stage, while the middle two rows have a dynamic issue stage. For the other two stages, the right most two columns hold machines with a dynamic execution stage, while the middle two columns are for dynamic retirement machines.

2.1 Static vs. Dynamic Behavior

To elaborate on the differences between static and dynamic behaviors we have collected some examples of each. There are at least two variations of static behavior and seven dynamic operational behaviors. The static variations are named lock-step and synchronized, while the dynamic are called reorder, dispose, expand, compact, split, merge, and decouple. Although an icon for some variations include a very small block diagram of a stage, the behavior is not necessarily dependent upon this implementation. Rather these examples are used to demonstrate how others have successfully implemented the various behaviors.

The two static behaviors are quite simple (Table 1). For a given static stage the input and the output should have the same number of instructions and be in the same order. For the lock-step variation, multiple instructions per cycle enter and exit a stage coupled together as a single word. This behavior is traditionally used throughout VLIW machines. In contrast a synchronized version maintains only the relative timing between otherwise independent instruction streams, in this case with queues. This implies that individual streams can be stalled when waiting for a value.

The first dynamic behavior, reorder, is a simple permutation within an instruction stream and requires some form of buffering to allow instructions to be reordered (Table 2). This behavior is useful in executing instructions out-of-order to compensate for data dependencies or to reorder instructions back into a precise order. The dispose dynamic variation is used primarily to support speculative processing in the form of either fetch, decode, or execution. For this behavior, a stage accepts instructions and conditionally throws away those associated with useless or mispredicted computation.

FD \ ER	SS	SD	DD	DS
SS	SSSS	SSSD	SSDD	SSDS
SD	SDSS	SDSD	SDDD	SDDS
DD	DDSS	DDSD	DDDD	DDDS
DS	DSSS	DSSD	DSDD	DSDS

Figure 3. 16-Fold Way Taxonomy.

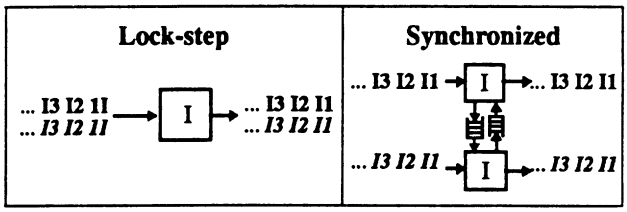

Table 1. Variations of Static Behavior.

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

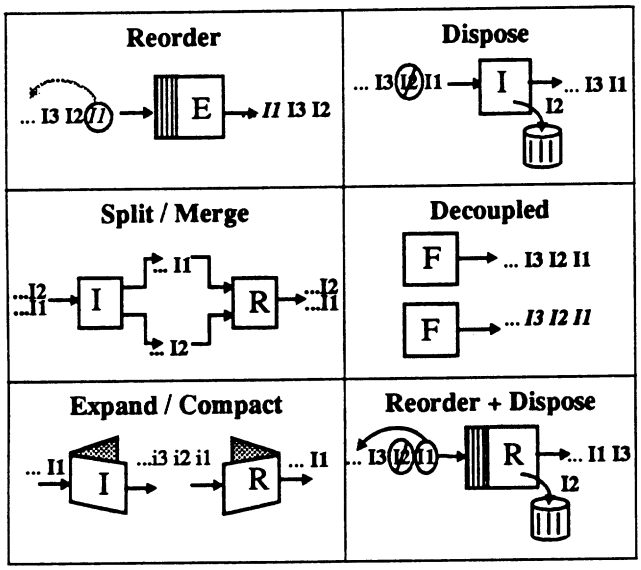

Table 2: Variations of Dynamic Behavior.

In the second row above, the split behavior accepts instructions and sorts them by class (e.g. integer, floating-point, or memory) potentially buffering them before issuing to decoupled streams. Alternatively, the merge behavior can be used either at the decode or retirement stage to merge separated streams back into a single instruction stream. The next component variation, decoupled, is used to denote the behavior of two completely autonomous streams. For example, in the fetch stage this requires separate program counters or at the issue and retirement stages this means there are separate register file images, one for each stream. The last variation can expand a single instruction (I1) into a set of internal operations (il, i2, and i3) which then perform specialized micro-operations possibly in parallel. Typically this occurs at the issue stage where instructions are decoded. Alternatively, a compact behavior reassembles these micro-operations back into the original form. The last box in Table 2 illustrates how two component behaviors can be combined into one stage. This example of a "reorder + dispose" variation can be used at the retire stage to support both speculative execution as well as reordering of instructions into a precise order.

Although these two static and seven dynamic component variations alone can theoretically lead to 9^4 possible machine configurations, not all variations are compatible. For example, reordering the instruction stream at the fetch stage, before decode, makes little sense unless we consider an associative matching of data to instructions as in a data flow execution model. But with different combinations of component behaviors, as in Table 2, there still exists a vast number of combinations. Therefore it is important to maintain a manageable number of classifications otherwise details can easily overwhelm the original intent of the taxonomy. Indeed, to its credit, Flynn's Taxonomy has survived so long because of its simplicity in describing complex computer systems with only four classifications. With a similar intent, but at a finer grain level, this paper sets forth a framework to understand the microparallelism found within the individual processors of a system with sixteen classes.

3.0 Machine Examples

With the taxonomy now defined, this section sorts a collection of machines that have been built or proposed by their closest behavioral FDER microparallel classification. Each machine's dynamic behavior is briefly described in the text that follows. A small block-diagram is also provided in Figure 4 which serves as a central reference point for the machines and is an expanded version of Figure 3.

3.1 The ELI-512: A SSSS Machine

The first classification instance to consider is the ELI-512 [Fisher83] which is a VLIW architecture. All four of its operational stages process, in a lock-step manner, a single stream of long-word instructions without altering the order. This results in an SSSS behavior classification. The obvious advantage of this style of machine is the simplicity of the hardware design which can lead to a very wide instruction width in excess of ten operations per word. However, a VLIW machine

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

does require considerable compile-time optimization to compensate for the rigid scheduling constraints, including aggressive loop unrolling [Lam88] and trace [Fisher81] or percolation [Nicolau85] scheduling.

3.2 The ZS-1: A SDSD Machine

After instructions are statically fetched by a single unit, various dynamic scheduling behaviors can be employed to enhance the performance of a processor. One such behavior is when the instruction stream is logically split. An instance of this category is the ZS-l [JSmith89] which is a split issue architecture. The static fetch stage of the ZS-1 processes a single stream of paired instructions. The issue stage then separates the instructions based upon their type. This serial-to-parallel transformation alters the relative ordering of instructions and is thus considered dynamic.

Because the now separated streams cooperate in performing a single task, there is a need to communicate between them. Thus in the ZS-1 the execute stage utilizes architectural queues to synchronize and pass data values between the streams. The advantage of this style of processing is that variable memory access is compensated when there is sufficient independent ALU and memory operations. This is because the streams have the freedom to breath by contracting or expanding, unlike the lock-step operation of an SSSS machine which can stall all of the stages. At the end of the ZS-1, the separated streams are then merged back into a single logical stream and so is a dynamic retirement stage.

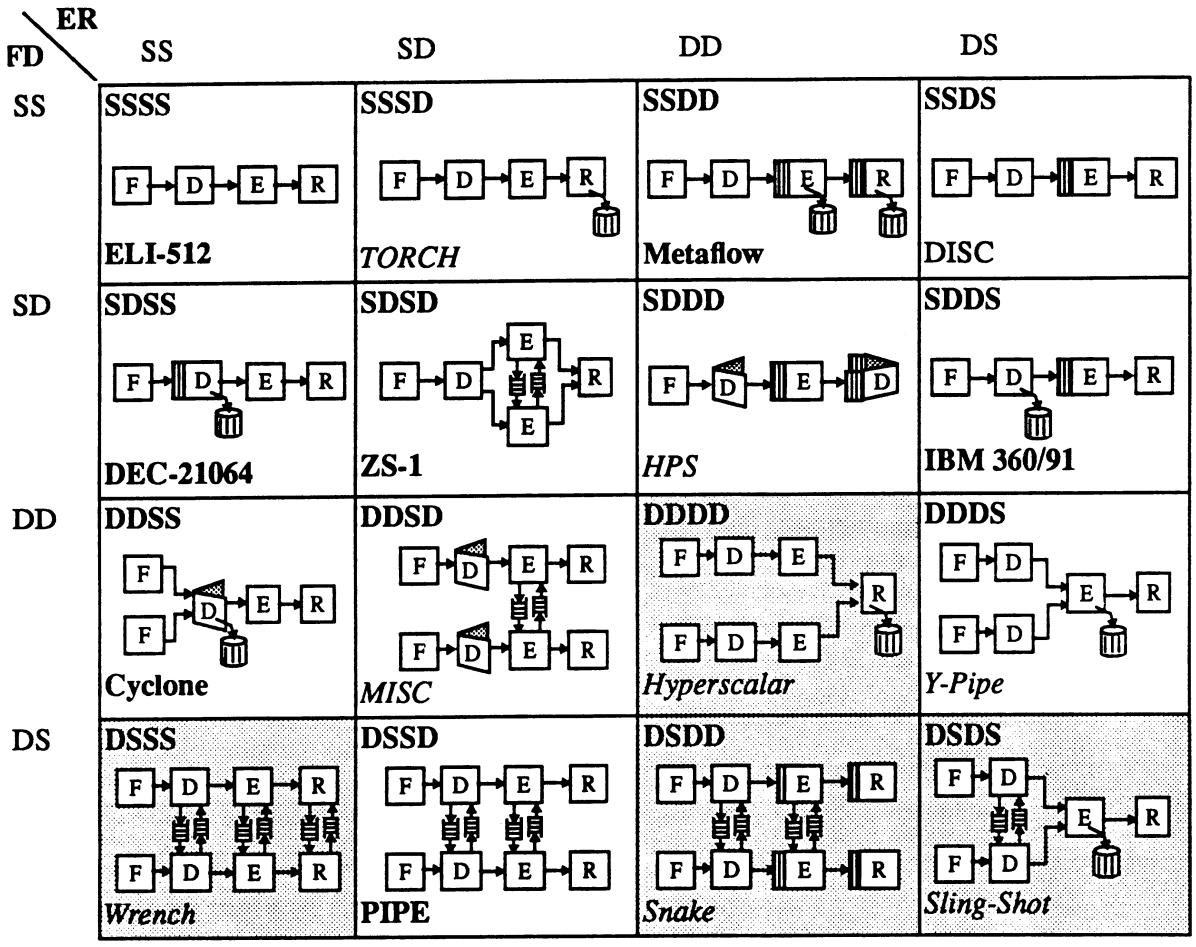

Figure 4. Microparallel Machine Instances.

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

3.3 The IBM 360/91: A SDDS Machine

The IBM 360/91 [Tomasulo67] is a significant machine because it is one of the most well known examples with the capability to dynamically reorder instructions at the execution stage. This processor fetches in static order, but disposes of speculatively fetched instructions at the decode stage and executes out-of-order with the help of reservation stations associated with each functional unit. This elaborate associative storage mechanism allows for partially decoded instruction to reside in a station until its operands appear on a common data bus that connects to the result registers of all the functional units. Once all operands of an instruction are gathered, it is executed and sent to the retire stage to be processed in static order disregarding the original fetch ordering.

3.4 The DEC-21064: A SDSS Machine

This classification contains many of the microparallel machines identified (Table 3). One instance of this dynamic-decode class is the DEC-21064 microprocessor [Dobb92] which has numerous features to support microparallelism. But, the only abstract dynamic behaviors to consider is its branch history table and scoreboard mechanism lumped together within the decode stage. The branch history table is similar in function to the IBM 360/91's decode stage which disposes of instructions that were speculatively fetched but are no longer of use, while the scoreboard mechanism is used to issue only instructions that have all of their operands ready. For correctly fetched and ready instructions, processing at the execute and retire stages is in-order or with a static behavior, unlike the dynamic-reordering execute stage of the IBM 360/91.

3.5 The DISC: A SSDS Machine

A rather obscure instance of this class is the DISC [Wang91]. This machine falls into this classification because after fetching and decoding in-order, its execute stage processes with a "forward and backward" routing network that recirculates instructions until they can be executed. In the DISC, data dependencies are specified *within* the instructions themselves in the form of cycle-count tags and are dynamically maintained by the individual functional units in a distributed manner. This means that a functional unit knows when it is "safe" to execute an instruction regardless of the state of other units, with the exception of variable memory latency. After instructions finally exit the execute stage they are retired without modification in a lock-step manner which represents a static behavior.

3.6 The TORCH: A SSSD Machine

When only a dynamic retirement stage disposes of instructions that have been statically processed, it influences the data state or programmer's view of the processor. This dynamic behavior can support the machine's ability to translate control dependencies into data dependencies, also known as speculative execution. An instance of this behavior is the TORCH machine [Msmith90, 92]. Its fetch, issue and execute stages are considered static in this study. The dynamic retire stage, however, disposes of instructions by selectively deleting useless instructions that are part of speculative computation. This dynamic behavior allows the compiler to enlarge basic blocks via branch prediction, boosting performance by exposing more instructions to a global scheduling algorithm, while at the same time providing an efficient run-time mechanism to delete the instructions.

3.7 The Metaflow: A SSDD Machine

The unfortunate effect of out-of-order execution is that a processor can no longer guarantee precise interrupts without reordering the side effects of instructions back into the original fetched order. To accomplish this task a reorder dynamic behavior can be placed at the last stage which retires instructions according to the order specified by the program counter giving the programmer the illusion of sequential execution along with the benefit of parallel processing. A machine that provides this behavior along with out-order execution is the Metaflow [Popescu 91]. This processor model actually implements a dynamic execute and retire stage in one unit called the DRIS which is a combination reservation station and reorder unit. Unlike the other machines, speculative computation can be in the process of executing and rearing so both these stages are augmented with a dispose variation to throw away instructions out of the DRIS.

3.8 The HPS: A SDDD Machine

One of the more ambitious efforts to exploit instruction-level parallelism is the High Performance Substrate (HPS). This processor model provides restricted data flow execution and over the years the HPS processor model has evolved to support a number of microparallel features for RISC and CISC instruction sets. In this case the VAX architecture is a good example [Patt86]. To support the multitude of addressing modes and interesting instruction semantics, the HPS can expand complex instructions into primitive "nodes," or micro operations issued to reservation stations called

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

"node tables." After executing in a dynamic execute stage, the micro-operations are compacted back into an image of the original instructions and reordered into a precise interrupt model.

The main advantage of this style of dynamic expand and compact behavior is code compatibility along with microparallel execution. The cost of this behavior is the considerable hardware to exploit both the intra-instruction and inter-instruction level parallelism. As such, the HPS is an extremely aggressive example of a static fetch and dynamic decode, execute and retire machine. While at the other end of the spectrum, a fully static VLIW processor typifies a relatively low cost alternative relying more upon the compiler to expose and exploit parallelism.

3.9 The PIPE: A DSSD Machine

The other main class of machines in this taxonomy involve some form of decoupled fetching. The first machine to implement a truly decoupled fetch stage was the PIPE [Goodman85] which illustrates a dynamic fetch and retirement taxon, or DSSD. It has the unique distinction of supporting two completely independent instruction synchronized at the decode and execute stages with architectural queues. The two streams are fetched from separate code regions and indexed by independent program counters. Each instruction stream has its own register file in the decode stage which is also where branch outcomes are passed between the streams to synchronize global branch points.

The execute stage has the same queue mechanism as the ZS-1, allowing ALU operations to overlap with memory operations. There are advantages however, to the PIPE architecture over the ZS-1 including the fact that completely separate streams can simplify the design of each instruction pipeline and that if one stream is stalled at the fetch stage the other is free to continue processing. The challenge for PIPE is how to efficiently balance the computation across multiple streams while keeping the communication between the cooperating pipelines to a minimum [Young85].

3.10 The Cyclone: A DDSS Machine

An interesting example of this classification is the Cyclone [Horst90], a combination of a decoupled fetch and merging decode stage. Lice the PIPE, this machine has two separate program counters and is thus considered a decoupled fetch machine. However, each PC fetches a portion of the same program with one PC periodically jumping ahead of the other. The issue stage dynamically pairs together the resulting two dynamic streams forming a single merged instruction stream sometimes throwing away instructions that were speculatively fetched. The instruction pairs are then processed in-order while in the execute and retire stages, resulting in a DDSS behavior. Not all pairs of instructions are allowed to execute in the Cyclone of course, because of the limited arrangement and number of functional units in the data path. But, by optimizing for the most frequently occurring instruction pairs, the Cyclone's control path is able to efficiently utilize the data path and other processor resources.

3.11 The MISC: A DDSD Machine

The machine instance of this class is the MISC [Tyson92] and is best described as a message-passing PIPE. Instead of passing computed predicates to the other pipelines, all information can be sent via the queues at the execute stage, including simple movement of data via direct queues. Although this will increase the bandwidth needed to synchronize the pipelines and distribute the control flow computation, it can simplify the implementation of communication between the streams. This is helpful for the MISC which extends the PIPE to four instruction streams instead of just two.

When the MISC is in a "vector" mode the machine resembles a DDSD behavior as shown in Figure 4. In this mode instructions are conditionally expanded depending on a vector length register while pipelines cooperate by passing data to cashes and other pipelines via queues. Although the MISC also has an alternate mode which issues instructions in a sentinel manner with issue stages synchronized (i.e. a DSSD), it has such a unique dynamic decoupled behavior that it is placed in this class.

3.12 The Y-Pipe: A DDDS Machine

The last variation on decoupled architectures separates the pipelines down to a dynamic execution and static retirement stage and is called the Y-Pipe [Knieser92] so named because it resembles the letter "Y." This DDDS behavior supports zero-cycle branch delays by duplicating the fetch and decode resources eagerly executing one instruction down both paths of a conditional branch. Execution then proceeds down only one path after the conditional is resolved. This can conceptually be done by deleting the unneeded instructions from the streams at the execute stage and only retiring useful computation.

3.13 Additional Machine Instances

The above machine instances are of course not alone in their respective classifications. Indeed, there are many

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

commercial and experimental processors that have microparallel capability. Table 3 below lists some of these processors along with the closest classification and a reference to their design.

Most of the machines have only one dynamic stage, with the exception of the OHMEGA, PLUM, RS/6000, SIMP and XIMD. The latter three are described below. Some of the more recent VLIW machines include the Multiflow, i860, WARP, and 19000. Although the LIFE processor is also called a VLIW, it is not a fully static processor because of its guarded execution which is similar to the TORCH's speculative execution support. For processors with two to four issue widths, modest amounts of dynamic behavior are provided with their branch prediction or out-of-order issue capability.

The IBM RS/6000 deserves special recognition because it was one of the first "superscalar" microprocessors to be introduced. Its considerable dynamic behavior is from a split+dispose decode, decoupled execute, and merge retire stage which corresponds to a SDDD classification (Figure 5). The decode stage disposes of mispredicted branches and also separates and buffers instructions in small queues. The decoupled execution stages (i.e. branch, floating-point, and integer units) are then free to process instructions independently, although they do not have to communicate with queues as in the ZS-1.

Another machine worth mentioning is the Single Instruction Stream / Multiple Instruction Pipelining processor, or SIMP. This machine fetches a block of four instructions at a time and distributes them to independent instruction pipelines which coordinate activities through a centralized register file. Within each pipeline, instructions are buffered and allowed to execute out-of-order. But only when all four instructions from the same fetched block are completed can the entire block be retired. This requires instructions to be gathered and reordered back into the original fetch order (i.e. merge and reorder). The SIMP also has speculative execution support that allows instructions to be executed and conditionally deleted which is why it is shown below with an additional dispose retirement behavior.

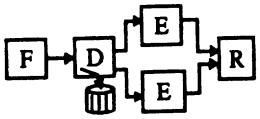

Figure 5. IBM RS/6000.

Table 3. Microparallel Machines.

Name	Class	Reference
CDC-6600	SDSS	[Thorton70]
Multiflow	SSSS	[Colwell87]
Precision Arch.	SDSS	[Jaffe92]
RS/6000	SDDD	[Balcoglu90]
i860	SSSS	[Inte1860]
i960	SDSS	[Inte1960]
iWARP	SSSS	[Cohn89]
LIFE	SSSD	[Slaven91]
MC68060	SDSS	[Gwen92]
MC88100	SDSS	[Case91]
OHMEGA	SDDS	[Nakajima91]
Pentium	SDSS	[Case93]
PLUM	SDDD	[Singhal89]
SIMP	SDDD	[Muakami89]
SuperSPARC	SDSS	[Case91]
Swordfish	SDSS	[Marko91]
T9000	SSSS	[Forsyth91]
XIMD	DDSS	[Wolfe91]

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.

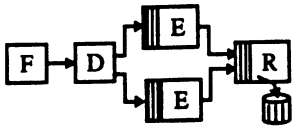

Figure 6. SIMP.

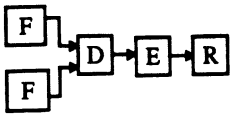

Figure 7.·XIMD.

The final established machine to consider is the XIMD which is a hybrid SIMD / MIMD machine. It contains separate fetch stages for independent instruction streams. But all of these streams are logically merged to access a central register file image, unlike the PIPE which has completely separate register files for each stream.

When the XIMD is executing in an SIMD mode all fetch units access the instruction memory with the same pattern, simulating the behavior of a VLIW. In a MIMD mode the fetch units diverge while following independent threads of computation. For either mode, however, the resultant dynamic instruction stream is merged and processed lock-step and in-order which is a static behavior for the execute and retire stages.

4.0 Postulated Machine Organizations

The previous sections provide established machine instances that fit a particular behavioral classification. This section fills the remaining unexplored "boxes" by combining various dynamic stages together to form four new machines and postulates their features.

4.1 The Wrench*: A new DSSS Machine

One problem with the PIPE is exceptions because a programmer must compensate for the autonomous retirement of separate instruction streams. Furthermore, if both pipelines are used to resolve an exception and restart execution, then both also need to context switch. To handle this we can synchronize the streams at the retirement stage with "guard" instructions that will check for traps and holdup the retirement of instructions until both guard instructions are reared at the same time. This transforms the decoupled retirement stage of the PIPE into a synchronized stage and therefore a DSSS behavior. This behavior guarantees points in the execution that the processor state of the streams will be synchronized simplifying the context switch operations.

4.2 The Sling-Shot*: A new DSDS Machine

An inadequacy of the Y-Pipe is that only one branch is eagerly processed by the duplicated resources. This simplifies the control and buffering of instructions but it does not utilize the full potential of speculative processing down two *threads* of computation or pursuing multiple-control flows [Lam92]. To enable this capacity, the decode stages of the Y-Pipe are synchronized which means they are free to perform branches down two different predicted paths and still communicate with a predicate queue holding the outcomes of comparisons. This means instructions can be fetched and decoded from two elongated, predicted paths with the execute stage skill disposing of the incorrectly predicted instructions. There is, however, the possibility that portions of both elongated paths are incorrect which will require rather sophisticated bookkeeping of the branches and probably increase the penalty of restarting the fetch stage down the correct path.

4.3 The Snake**: A Dew DSDD Machine

This machine is envisioned to be a decoupled fetch architecture similar to the PIPE with an execute stage that can reorder instructions in each stream. The justification for this particular setup is that the PIPE stops the access stream on cache misses. Unfortunately, this can stop dependent instructions in the execute stream. If there are multiple architectured queues however, a miss can then be bypassed with other independent operations, further overlapping the miss processing with memory service or even other memory access. This should increase the machine's tolerance to variable memory latency. Additionally, this machine needs to rearrange the instructions into a consistent order after the dynamic execute stage and so has a decoupled, reorder-retirement stage.

4.4 The Hyperscalar: A new DDDD Machine

The final machine to consider is a decidedly hardware intensive processor capable of performing speculative computation through the execution stage, but without the compilation techniques of the TORCH. Thus the object code can have serial semantics and the processor can dynamically allocate a different pipeline of the ma-

* Fixes a PIPE.

* Resembles a sling-slot in Table 3.

** Clears clogged plumbing.

chine for every possible outcome of a branch. The illustration in Table 3 only shows two pipelines, but conceptually there can be any number of these pipelines operating independently, replicating execution at branch points and processing all paths until the retire stage where only one path is committed to the processor state. Certainly this is unrealizable as a single-chip microprocessor, but the intent is clearly to burden the hardware with the task of dynamically exploiting instruction-level parallelism while still executing existing object code.

5.0 Conclusion

In this paper we propose a two valued, four variable microparallel taxonomy, called the 16-Fold Way, which results in 16 different processor microarchitecture classifications. We categorized established machines into twelve of these 16 different classes. Each machine is defined in terms of its behavior during the four stages of instruction processing; fetch, decode, execute and retire. The behavior can be one of nine component behaviors; lock-step, synchronize, reorder, dispose, expand, compact, split, merge, and decouple. These behaviors are broadly clarified as either static or dynamic with some machines containing a combination of components.

From Figure 4 we can see that the fetch stage is either a single unit or a pair of units. In the first case this stage fetches a single stream of instructions, occasionally fetching down one predicted path. In the second case it can be a decoupled (dynamic) stage that either fetches multiple cooperating streams, or eagerly fetches multiple paths within a single program. At the issue stage the diversity of dynamic behavior varies the most, consisting of all static and dynamic behaviors except compacting. For the execute stage, the figure reveals that for established machine categories (i.e. white boxes) the most common dynamic behavior is reordering. For the last stage, rearing of instructions generally requires some behavior to support either speculative execution by disposal, or precise interrupts with reordering or synchronization.

This microparallel taxonomy illustrates the great diversity of machines capable of processing instructions in parallel. Each classification holds an abundant variety of processor organizations that have yet to be investigated. This paper has outlined examples for each of the sixteen classifications but none are meant to be representative. Rather they are a first attempt at understanding what capabilities these classes contain. Thus, the intent of this taxonomy is to provide a framework for postulating new designs. As new microparallel machines are proposed, this framework can be used to systematically compare and evaluate these designs.

Acknowledgments

This research was funded by ARPA under contract number J-FBI-91-194.

References

[**Bakoglu90**] H. Bakoglu and T. Whiteside, “RISC System 6000 Hardware Overview,” *IRM RISC System Technology*, 1990.

[**Case91**] B. Case, “Superscalar Techniques: SuperSPARC vs. 88110,” Microprocessor Report, Vol. 5, No. 22, Dec. 4th, 1991.

[**Case93**] B. Case, “Intel Reveals Pentium Implementation,” Microprocessor Report, Vol. 7, No. 4, Mar. 29th, 1993.

[**Chang91**] P. Chang, et al., “IMPACT: An Architectural Framework for Multi-Instruction-Issue Processors,” *Proc. 18th Ann. Int’l Symp. Computer Architecture*, 1991.

[**Cohn89**] Robert Cohn, et al., "Architecture and Compiler Tradeoffs for a Long Direction Microprocessor,” *Proc. 3rd Conf. Architectural Support for Programming Languages and Operating Systems*, 1989.

[**Colwel187**] R. P. Colwell et al., “A VLIW Architecture for a Trace Scheduling Compiler,” *2nd Int’l Conf. Architectural Support for Programming Languages and Operating Systems*, 1987.

[Dobb92] D. W. Dobberpuhl, et al., “A 200 MHz 64-b Dual-Issue CMOS Microprocessor,” *IEEE J. Solid-State Circuits*, Vol. 27, No. 11, Nov. 1992.

[**Fisher81**] J. A. Fisher, “Trace Scheduling: A Technique for Global Microcode Compaction,” *IEEE Trans. Computers*, Vol. C-30, N0. 7, July 1981.

[Fisher83] J. A. Fisher, “Very Long Instruction Word Architectures and the ELI-512,” *Proc. 10th Ann. Int’l Symp. Computer Architecture*, 1983.

[Fisher91] J. A. Fisher and B. R. Rau, “Instruction-Level Parallel Processing,” *Science*, Sept. 13th 1991.

[**Flynn72**] M. J. Flynn, “Some Computer Organization and Their Effectiveness,” *IEEE Trans. Computer*, vol. c-21, no. 9, Sept. 1972.

[Forsyth91] R. Forsyth Bob Krysiak, and Roger Shepard, “T9000—Superscalar Transputer,” *Hot Chips 111 Presentation*, Aug. 1991.

[**Goodman85**] J. R. Goodman et al., “PIPE: A VLSI Decoupled Architecture” *Proc. 12th Ann. Int’l Symp. Computer Architecture*, 1985.

[**Gwen91**] L. Gwennap, “Motorola Details Plan to Extend 68K Line,” Microprocessor Report, Vol. 6, No. 15, Nov. 18th 1992.

[Horst90] R. W. Horst, R.L. Harris, and RL. Jardine, “Multiple Instruction Issue in the NonStop Cyclone Processor,” *Proc. 17th Ann. Int’l Symp. Computer Architecture*, 1990.

[Intel860] “64 Bit Microprocessor Programmers Reference Manual,” Intel Corporation, Mt. Prospect, Ill., 1990.

[**Intel960**] “80960CA User’s Manual,” Intel Corporation, Santa Clara Calif., 1989.

[Jaffe92] W. Jaffe, B. Miller, and J. Yetter “A 200 MFLOP Precision Architecture Processor,” *Hot Chips IV Presentation*, 1992.

[JSmtth89] J. E. Smith, “Dynamic Instruction Scheduling and the Astronautics ZS-1,” *Computer*, pp. 21-35, June 1989.

[Jouppi91] N. P. Jouppi and D. W. Wall, "Available Instruction-Level Parallelism for Superscalar and Superpiplined Machines," *Proc. 3rd Conf. Architectural Support for Programming Languages and Operating Systems*, 1989.

[Kniser92] M. Knieser and C. Papachristou, "Y-Pipe: A Conditional Branching Scheme Without Pipeline Delays," *Proc. 25th Ann. Int'l Symp. Microarchitecture*, 1992.

[**Kuga91**] M. Kuga, K. Murakami, and S. Tomita, "DSNS (Dynamically -hazard-resolved, Statically-code-scheduled, Nonuniform Superscalar): Yet Another Superscalar Processor Architecture," *Computer Architecture News*, vol. 19, no. 4, June 1991.

[**Lam88**] M. Lam, "Software Pipelining: An Effective Scheduling Technique far VLIW Machines," *ACM SIGPLAN '88 Conf. Programming Language Design and Implementation*, 1988.

[**Lam92**] M. Lam and R. P. Wilson, "Limits of Control Flow on Parallelism," *Proc. 19th Ann. Int'l Symp. Computer Architecture*, 1992.

[**Marko91**] R. Marko and M. Beck, "National's Swordfish A Superscalar with DSP," *Hot Chips 111 Presentation*, 1991.

[MSmith90] M. Smith, M. Lam, and M. Horowitz, "Boosting Beyond Static Scheduling in a Superscalar Processor," *Proc. 17th Ann. Int'l Symp. Computer Architecture*, 1990.

[**MSmith92**] M. Smith, M. Horowitz, and M. Loin, "Effficient Superscalar Performance Through Boosting," *Proc. 5th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 1992.

[Murakami89] K. Murakami, et al., "SIMP (Single Instruction stream/ Multiple instruction Pipelining): A novel High-Speed Single-Processor Architecture,'" *Proc. 16th Ann. Int'l Symp. Computer Architecture*, 1989.

[**Nakajima91**] M. Nakajima, et al., "OHMEGA: A VLSI Superscalar Processor Architecture for Numerical Applications" *Proc. 18th Ann. Int'l Symp. Computer Architecture*, 1991.

[**Nicolau85**] A. Nicolau, "Percolation Scheduling: A Parallel Compilation Technique'," CS Technical Report TR 8S-678, Cornell University, Ithaca, N.Y., 1985.

[**Patt86**] Y. Patt et al., "Run-Time Generation of HPS Microinstructions from a VAX Instruction Stream," *Micro 19 Workshop*, 1986.

[**Popescu91**] V. Popescu, "The Metaflow Architecture," *IEEE Micro*, June 1991.

[**Singhal89**] A. Singhal, "A High Performance Prolog Processor with Multiple Functional Units," *Proc. 16th Ann. Int'l Symp. Computer Architecture*, 1989.

[**Slaven91**] G. A. Slavenburg, et al., "The LIFE Family of High Performance Single Chip VLIWs," *Hot Chips 111 Presentation*, 1991.

[Thorton70] J. E. Thorton, "Design of a Computer-The Control Data 6600," *Scott, Foresman and Co.*, Glenview IL 1970.

[Tomasulo67] R.M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM J. Research Development*, vol 11. pp.25-33, Jan. 1967.

[**Tyson92**] G. Tyson, M. Farrens, and A. Pleszkcun, "MISC: A Multiple Instruction Stream Computer," *Proc. 25th Ann. Int'l Symp. Microarchitecture*, 1992.

[**Wang91**] L. Wang and C. Wu, "Distributed Instruction Set Computer Architecture," *IEEE Trans. Computers*, vol. 40, no. 8, Aug. 1991.

[**Wolfe91**] A. Wolfe and J. P. Shen, "A Variable Instruction Stream Extension to the VLIW Architecture," *Proc. 4th Conf. Architectural Support for Programming Languages and Operating Systems*, 1991.

[**Young85**] H. Young, "Evaluation of a Decoupled Computer Architecture and the Design of a Vector Extension," Computer Sciences Technical Report #603, University of Wisconsin-Madison, July 1985.

Originally published in *Proc.26th Ann. Int'l Symp. Microarchitecture*, 1993, pp. 60–69.