NISC Application and Advantages

Daniel D. Gajski
Mehrdad Reshadi
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
{gajski, reshadi}@cecs.uci.edu

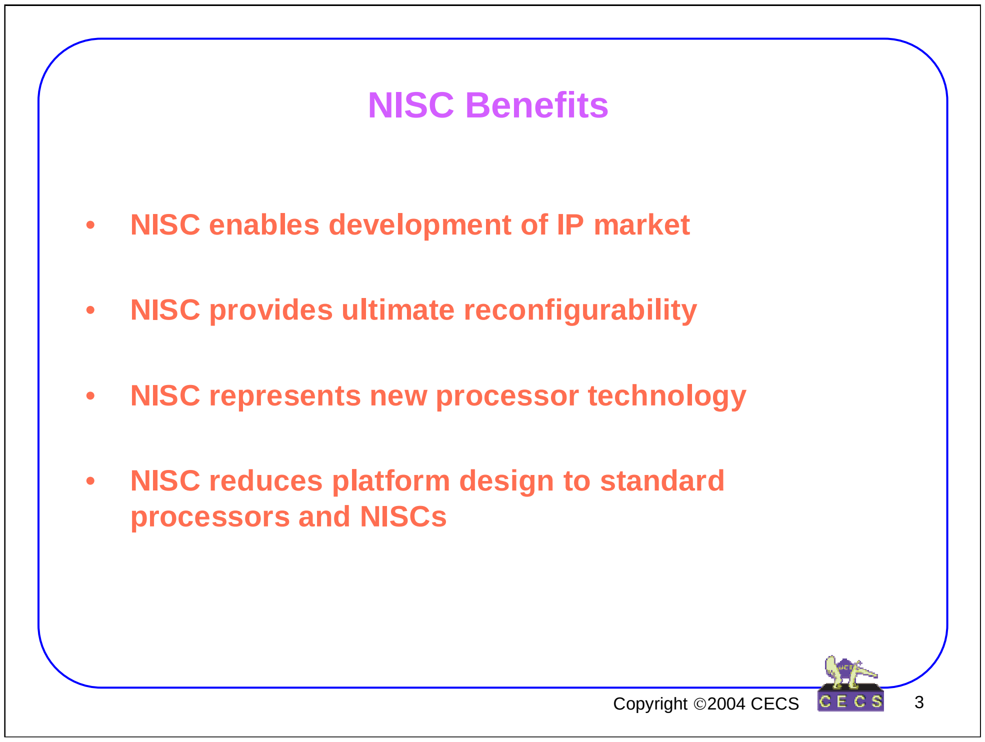
CECS Technical Report 04-12
May, 2004

NISC Application and Advantages

Daniel D. Gajski
Mehrdad Reshadi
Center for Embedded Computer Systems
University of California, Irvine
Irvine, CA 92697-3425, USA
{gajski, reshadi}@cecs.uci.edu

Introduction

With complexities of Systems-on-Chip rising almost daily, the design community has been searching for new methodology that can handle given complexities with increased productivity and decreased times-to-market. The obvious solution that comes to mind is increasing levels of abstraction, or in other words, increasing the size of the basic building blocks. However, it is not clear how many of these building blocks we need and what these basic blocks should be. Obviously, the necessary building blocks are processors and memories. One interesting question is: “Are they sufficient?”. The other interesting question is: “How many types of processors and memories do we really need?”. In this report we try to answer both of these questions and argue that the No-instruction-set computer (NISC) is a single, necessary and sufficient processor component for design of any digital system.



NISC Benefits

- NISC enables development of IP market
- NISC provides ultimate reconfigurability
- NISC represents new processor technology
- NISC reduces platform design to standard processors and NISCs

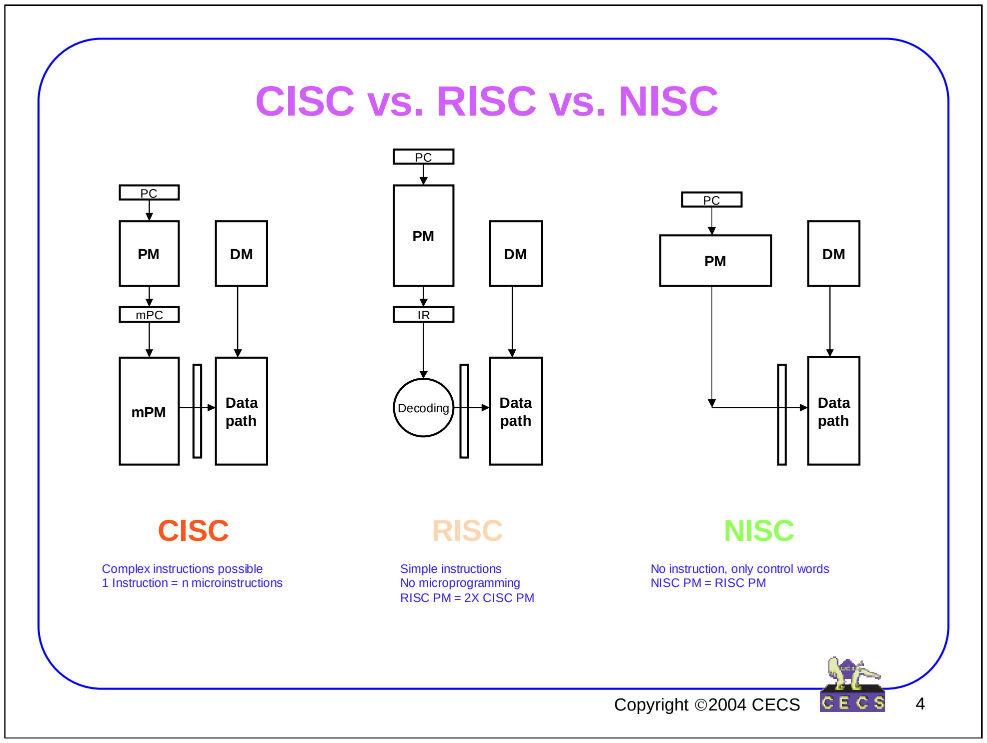
NISC Benefits

NISC technology is an enabler for the IP market. It provides a common microarchitecture, compiler and simulator for all IPs. Each IP can be implemented as a NISC.

NISC is an ultimate reconfigurable component since its microarchitecture is defined by connectivity of RTL components such as registers, register files, memories, ALUs, shifters, buses and others. Therefore, any NISC can be reconfigured at any time.

NISC represents a new processor technology since it eliminates the instruction set, the last interpretation step between the programming language and the hardware that executes it.

Since any component can be implemented as NISC, any platform can be built from standard processors, memories and NISCs.



CISC vs. RISC vs. NISC

CISC

Complex instructions possible
1 Instruction = n microinstructions

RISC

Simple instructions
No microprogramming
RISC PM = 2X CISC PM

NISC

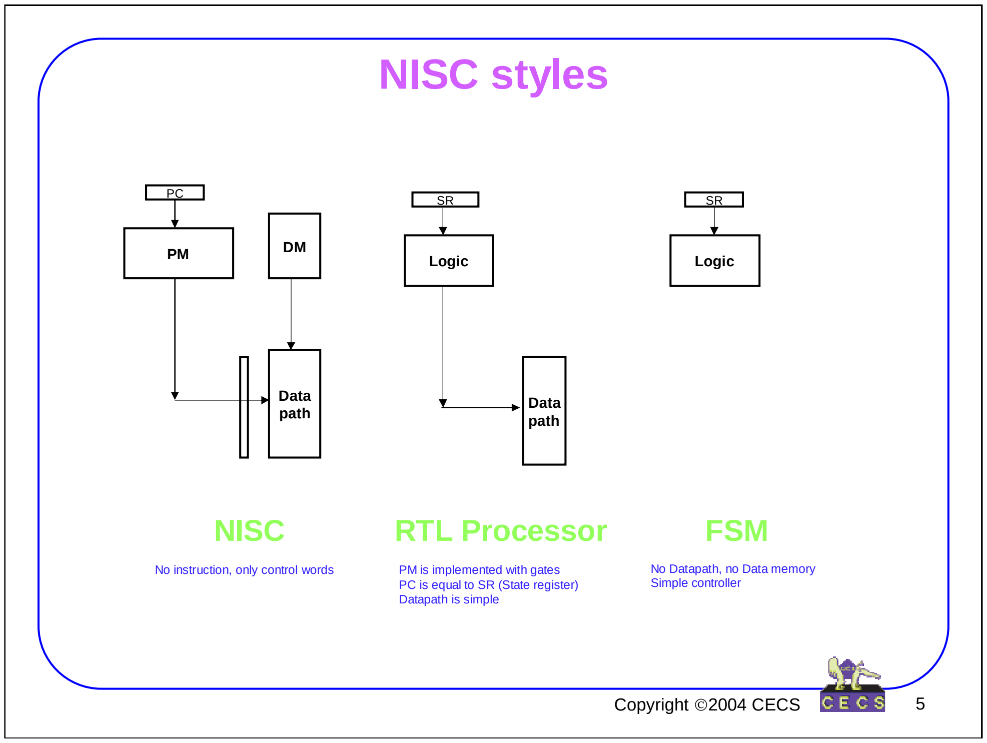
No instruction, only control words
NISC PM = RISC PM

4

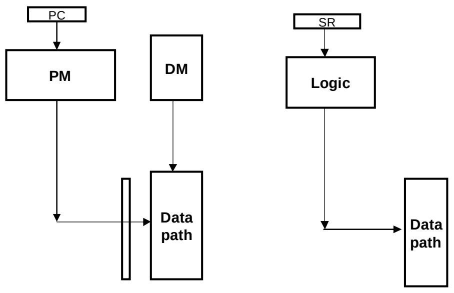
History of processor architecture

The evolution of the processor architecture can be divided into three phases.

1. Complex-instruction-set computer (CISC) was popular in 1970s. Since the program memory (PM) was slow, designers tried to improve performance by constructing complex instructions. Each complex instruction took several clock cycles, with Datapath control words for each clock cycle were stored in a much faster micro program memory (mPM). The concept of micro programming allowed for emulation of any instruction set and construction of specialized instructions, while speeding up the execution. Unfortunately, micro programming did not allow for efficient pipelining of the Datapath.
2. Reduced-instruction-set computer (RISC) became popular in late 1980s by eliminating complex instructions and the mPM. All instructions in a RISC are simple and execute in one clock cycle allowing Datapath to be efficiently pipelined in 4-8 pipelined stages. The mPM was replaced with decoding stage, that followed the instruction fetch from PM. Since instructions are simpler, a RISC needs approximately two instructions for each complex instruction and, therefore, the size of the PM is doubled. However, the Fetch-Decode-Execute-Store pipeline of the whole processor improved the execution speed several times.
3. The No-instruction-set computer (NISC), introduced here, completely removes the decode stage and stores the control word in the PM. Since control words are 2-3 times wider than instructions the PM increases in width by 2-3 times. Fortunately, each control word can execute 2-3 RISC instruction. Therefore, NISC PM = RISC PM. Furthermore each NISC is parametrizable and reconfigurable, which allows for very fine tuning to any application and performance.



NISC styles



NISC

No instruction, only control words

RTL Processor

PM is implemented with gates
PC is equal to SR (State register)
Datapath is simple

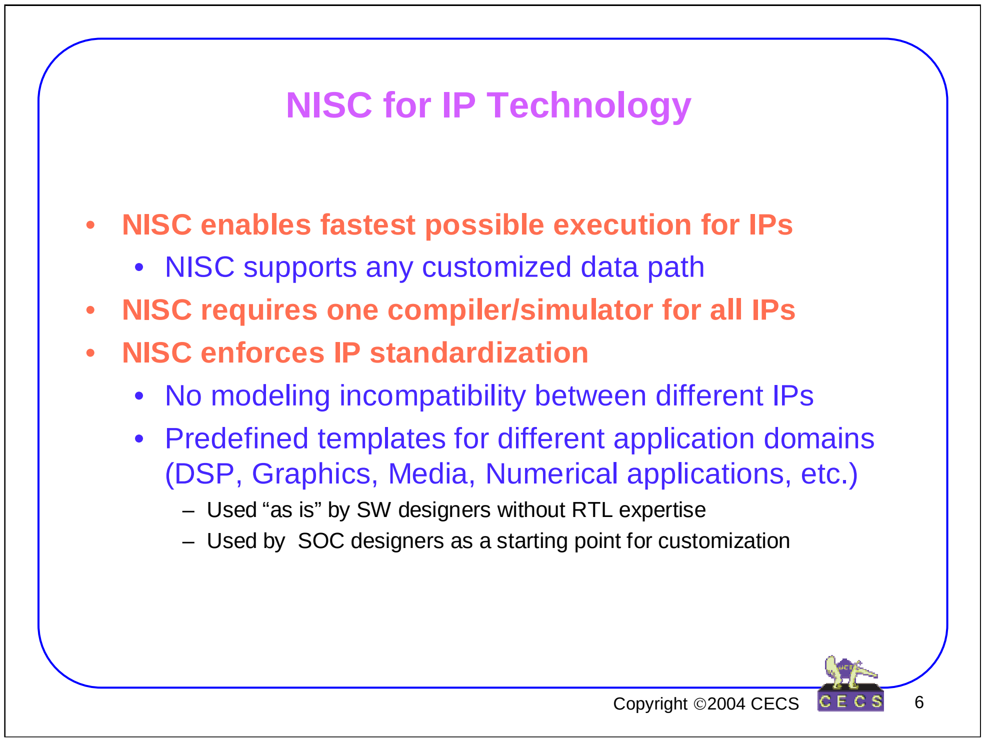
FSM

No Datapath, no Data memory
Simple controller

CECS 5

NISC styles

1. The Datapath in each NISC is parametrizable in terms of number of storage and functional units as well as in terms of number of buses. Therefore, NISC datapath can be statically or dynamically reconfigured as needed. NISC compiler will generate control words for each type of Datapath.
2. If the PC is replaced with a State register (SR) and PM is implemented with logic gates that determine the next state and control a simple Datapath in each state then such a NISC is usually called RTL processor. In a RTL processor the Data memory is very small or non existent.
3. If the Datapath is completely removed then such oversimplified NISC is called Finite-state-machine (FSM). The Logic defines the next state and output signals. Such FSM is used for some simple controllers. Present CAD tools are capable of synthesizing FSMs and some very simple RTL processors.



NISC for IP Technology

- NISC enables fastest possible execution for IPs
  - NISC supports any customized data path
- NISC requires one compiler/simulator for all IPs
- NISC enforces IP standardization
  - No modeling incompatibility between different IPs
  - Predefined templates for different application domains (DSP, Graphics, Media, Numerical applications, etc.)
    - Used "as is" by SW designers without RTL expertise
    - Used by SOC designers as a starting point for customization

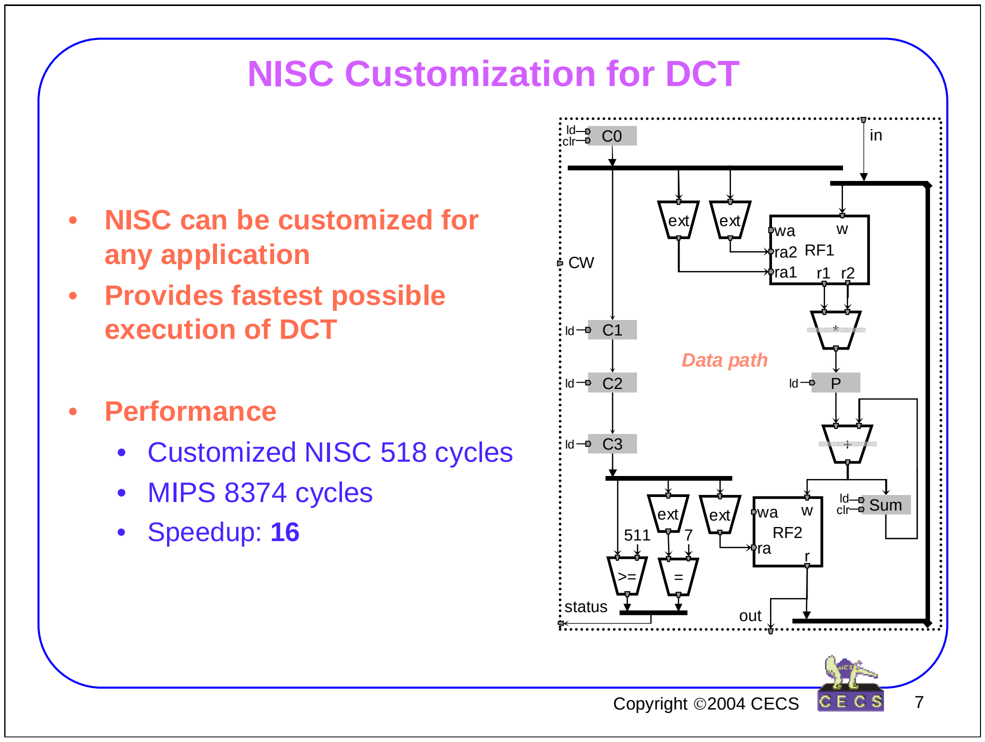
NISC for IP Technology

Since NISC supports application tuning through microarchitecture customization it can, thus, achieve the highest possible performance. For example, NISC customized for DCT executes the whole DCT in only 518 clock cycles while a MIPS spends more than 8000 clock cycles to finish the same code.

NISC model defines its data path in terms of register transfers. Such a model is used by the compiler to map any application to a given data path. Similarly, the simulator uses the model to simulate any NISC. In this way, only one compiler and simulator are sufficient for any NISC style design.

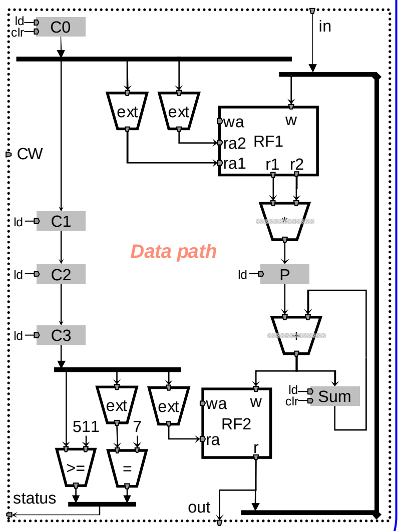
Today, IPs are offered on different levels of abstractions with incompatible models which prevent the composition of several IPs together in a design. In cases other than processor cores, mapping the application on the IP must be done manually. For programmable cores, two different models for compilation and simulation must be provided. On the other hand, one NISC models is sufficient for both compilation and accurate simulation. The NISC concept also enables us to automatically map any application to any NISC model. In other words, by using the NISC model of IPs, the users can easily combine many IPs together and avoid many of the manual procedures in the design flow.

By generating NISC templates for each category of applications, users with no deep hardware knowledge can put the NSIC models together and have the NISC compiler use them. Such templates can also be used by designers as a starting point for generating a data path for a specific application.



NISC Customization for DCT

- **NISC can be customized for any application**
- **Provides fastest possible execution of DCT**
- **Performance**
  - Customized NISC 518 cycles
  - MIPS 8374 cycles
  - Speedup: **16**

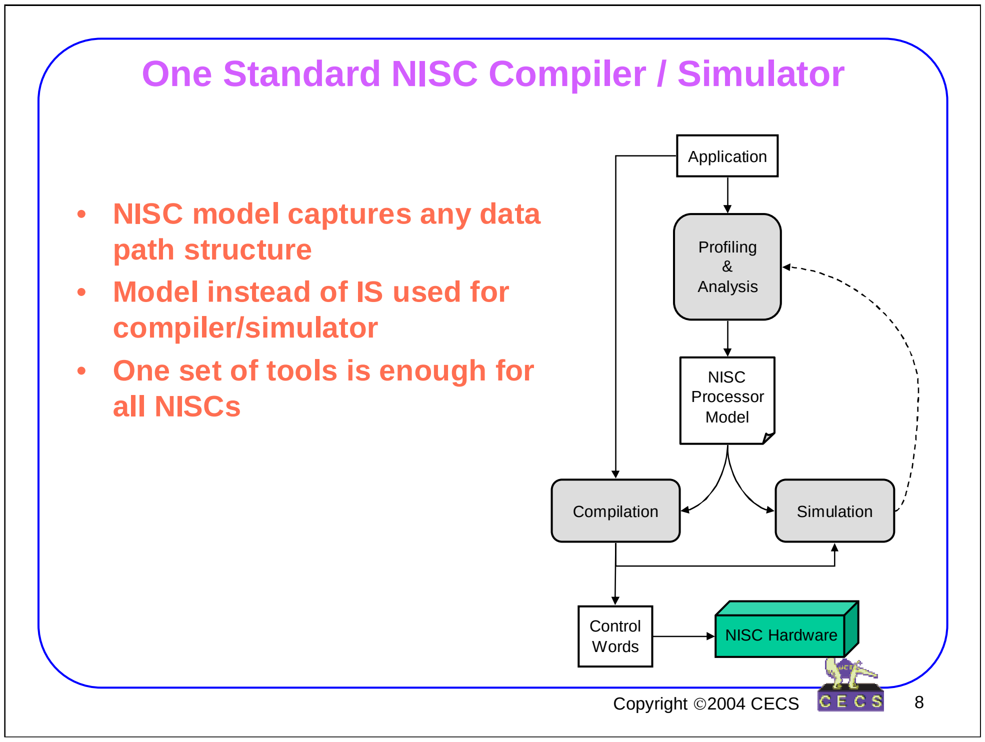


7

Custom NISC for DCT (Discrete Cosine Transform)

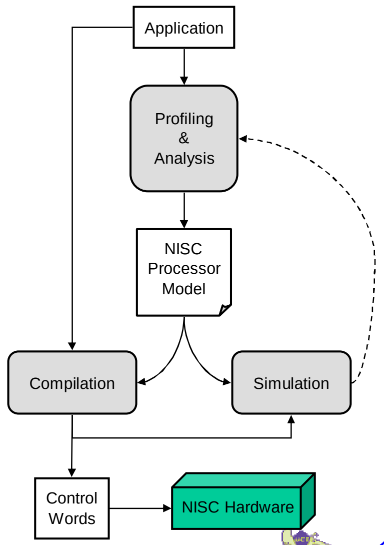
The above data path shows the most efficient way of implementing the DCT. It consists of multiply-add pipeline and two register files that store data frame and constant matrices. In parallel with pipelined DCT computation, the datapath increments the address registers for each register file. Also, the detection of loop boundary is done in parallel.

Using synthesis algorithms rather than traditional ones, the NISC compiler can map the DCT code to this customized data path. While the same DCT code takes 8374 clock cycles on a MIPS, the code will finish in only 518 clock cycles on this customized data path. In this way a speedup of 16 is achieved. The clock cycle is the same in both cases, however, since the controller of NISC is simplified, its clock speed can go even higher than that of MIPS.



One Standard NISC Compiler / Simulator

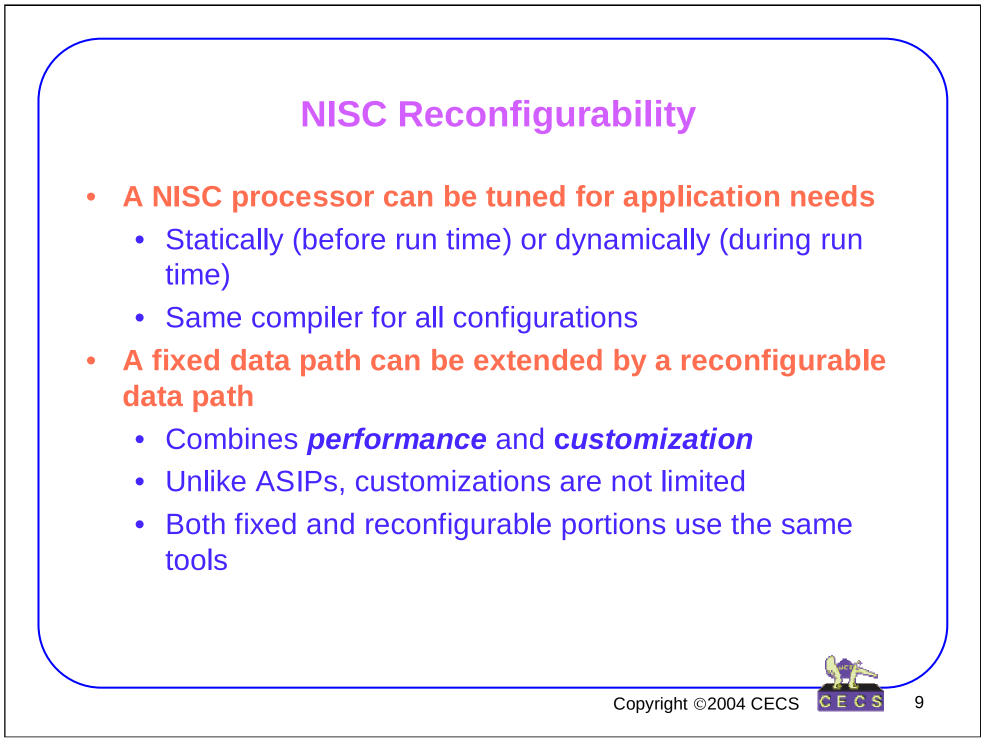
- NISC model captures any data path structure
- Model instead of IS used for compiler/simulator
- One set of tools is enough for all NISCs



8

NISC Design Flow

The picture shows the typical process of designing the best possible NISC for an application. In this process, the application is first profiled and analyzed to extract its features. Based on the results, a NISC is designed and then defined by the NISC model. The compiler uses this model to translate the application to the corresponding control words for the target hardware. Each control word defines the behavior of the architecture in one clock cycle. The simulator simulates the output of the compiler on the input NISC model. The simulation results can be used for further analysis and refinement of the data path. Since the compiler and simulator use the NISC model rather than assuming a fixed instruction set, only one compiler/simulator is sufficient for any NISC design. Specially for the compiler, this is achieved by removing the instruction-set interface and using synthesis oriented algorithms in the compiler to generate register transfers.



NISC Reconfigurability

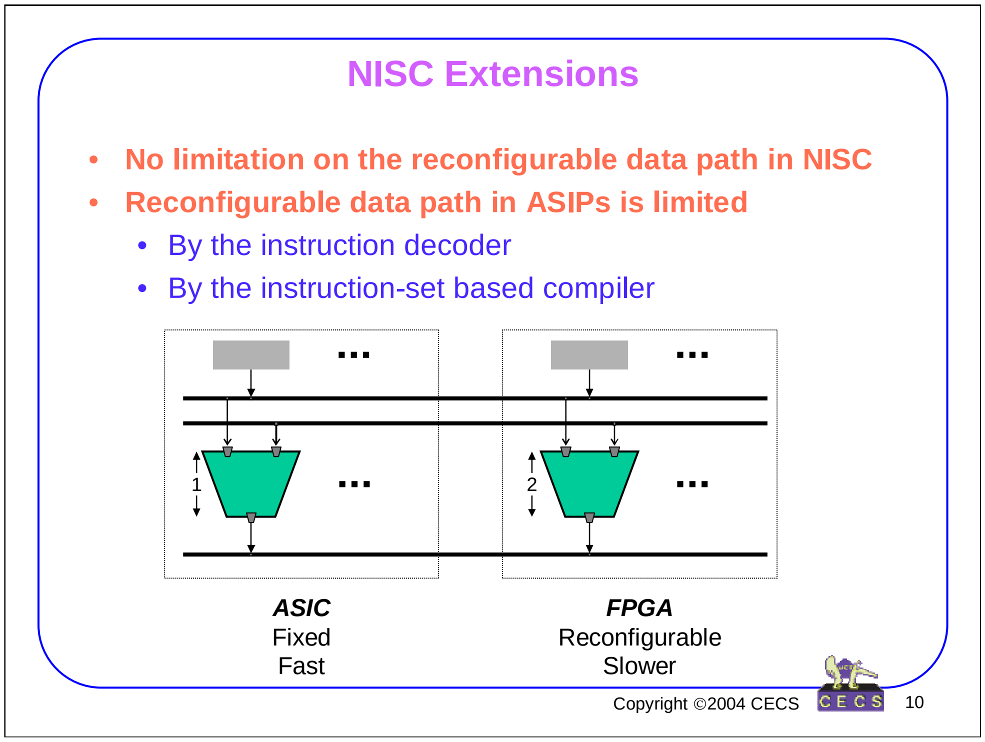
- A NISC processor can be tuned for application needs
  - Statically (before run time) or dynamically (during run time)
  - Same compiler for all configurations
- A fixed data path can be extended by a reconfigurable data path
  - Combines ***performance*** and ***customization***
  - Unlike ASIPs, customizations are not limited
  - Both fixed and reconfigurable portions use the same tools

NISC and Reconfigurability

A NISC processor can be implemented in any technology. Since the a NISC is defined by the structure of its components and not the instruction set it is inherently reconfigurable. Therefore, it is a perfect match for reconfigurable technologies. A reconfigurable fabric such as an FPGA can be reconfigured statically before run time or dynamically during run time to tune a NISC to a specific application.

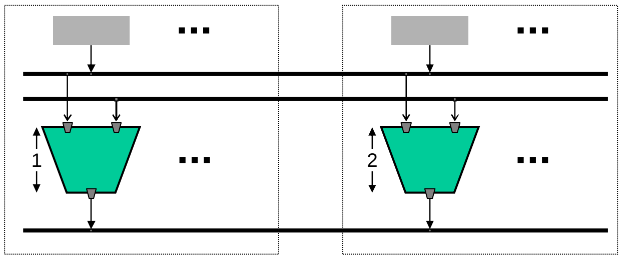
The same compiler/simulator are applicable for any configuration.

Components implemented in a reconfigurable circuits are usually slower than their ASIC counterparts. To combine a high performance circuit with a customizable one, a fixed data path can be extended by a reconfigurable one. Again, both data paths are handled by the same set of NISC tools. Unlike ASIPS, there is no limitation on the size or structure of reconfigurable data path in the NISC approach.



NISC Extensions

- No limitation on the reconfigurable data path in NISC
- Reconfigurable data path in ASIPs is limited
  - By the instruction decoder
  - By the instruction-set based compiler



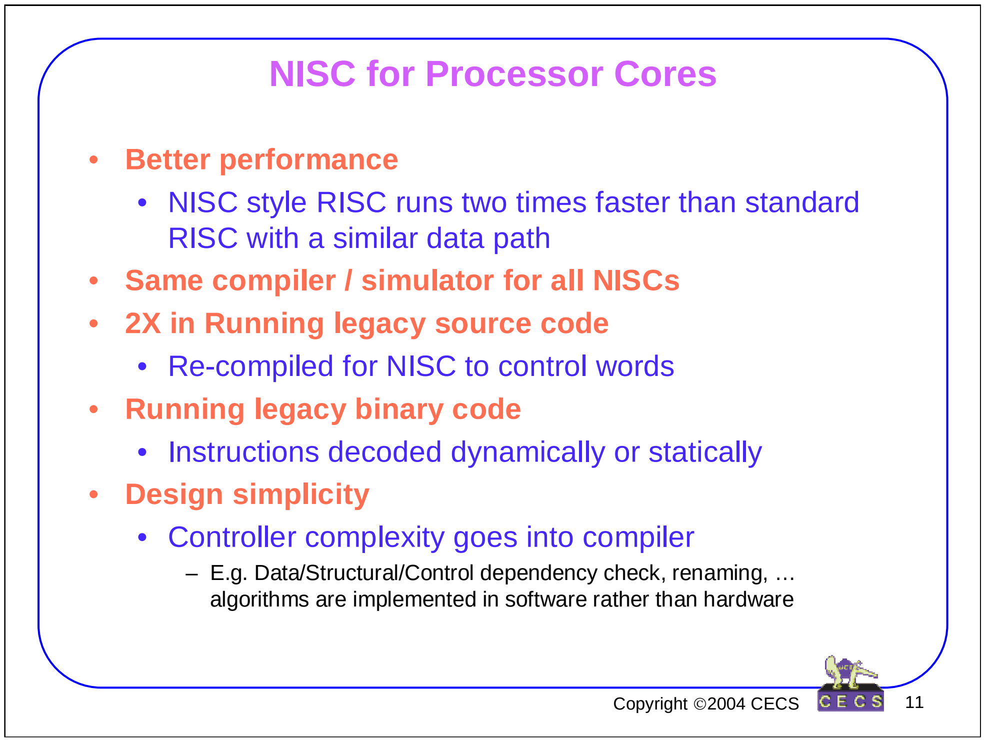
ASIC
Fixed
Fast

FPGA
Reconfigurable
Slower

Extending a NISC Data Path

In ASIPs, the instruction set of a processor can be expanded by a reconfigurable logic. However, not all possible functionalities can be supported by the instruction decoder and the compiler. This will limit the use of reconfigurable logic and the possible customizations for a specific application.

In NISC, a fixed data path can be extended by an additional reconfigurable data path with no limitations. As shown in the above figure, the components in the reconfigurable fabric may be slower and therefore the compiler must decide when to use them. Since the compiler can handle multi-cycle units, there is no limitation on the timing of the components.



NISC for Processor Cores

- **Better performance**
  - NISC style RISC runs two times faster than standard RISC with a similar data path
- **Same compiler / simulator for all NISCs**
- **2X in Running legacy source code**
  - Re-compiled for NISC to control words
- **Running legacy binary code**
  - Instructions decoded dynamically or statically
- **Design simplicity**
  - Controller complexity goes into compiler
    - E.g. Data/Structural/Control dependency check, renaming, … algorithms are implemented in software rather than hardware

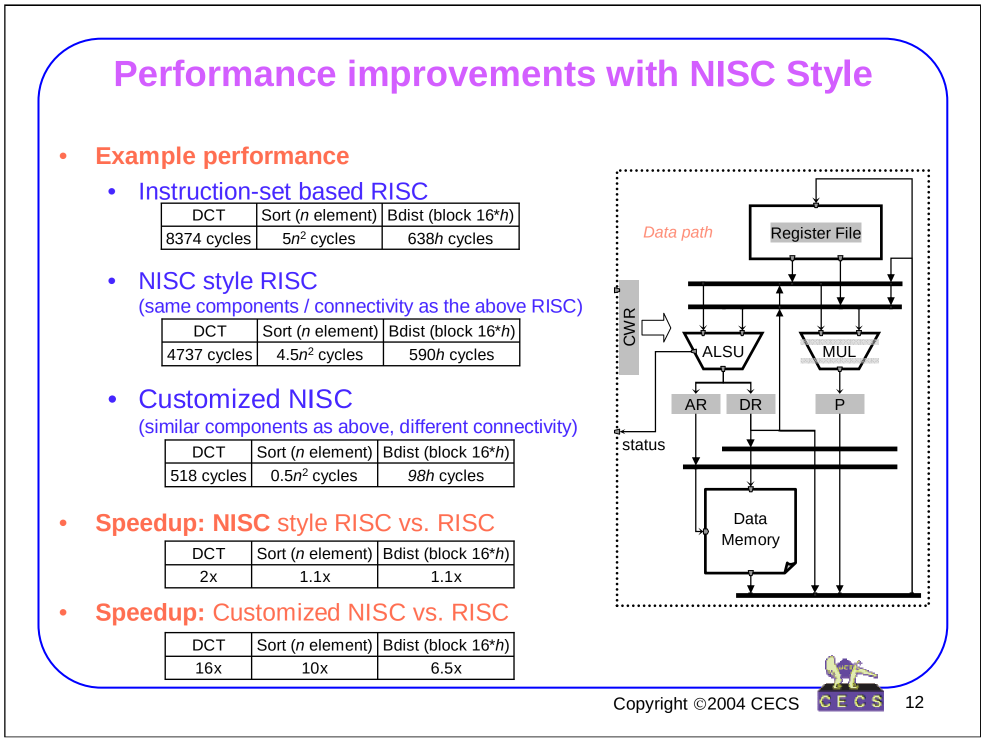
NISC Style Processors

Since a NISC style processor is defined on register transfer level, it offers more parallelism and more code compaction. In other words, a NISC style processor delivers a better performance than an instruction-set based processor with the same data path.

Again the NISC compiler and simulator can be used for any NISC processor.

It is also possible to run any legacy application on a NISC processor. A legacy source code can be compiled on a NISC processor and will run as fast as the NISC microarchitecture allows. On the other hand the binary instructions in a legacy binary code, can be translated to proper control words that executes each instruction. This translation is in fact simple table lookup that maps an instruction to its corresponding control words.

Additionally, designing NISC style processors is much simpler than traditional ones. The complex algorithms for detecting data hazards, structural hazards, control hazards, etc., are moved into compiler which makes the hardware controller much simpler.



Performance improvements with NISC Style

- Example performance
  - Instruction-set based RISC

DCT	Sort (n element)	Bdist (block 16*h)
8374 cycles	$5n^2$ cycles	638h cycles

  - NISC style RISC
    (same components / connectivity as the above RISC)

DCT	Sort (n element)	Bdist (block 16*h)
4737 cycles	$4.5n^2$ cycles	590h cycles

  - Customized NISC
    (similar components as above, different connectivity)

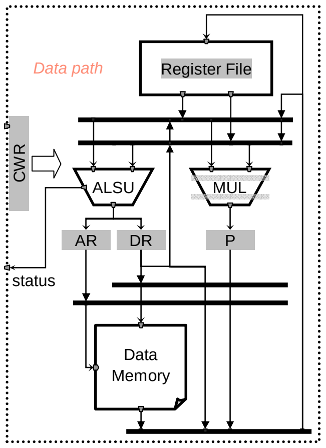
DCT	Sort (n element)	Bdist (block 16*h)
518 cycles	$0.5n^2$ cycles	98h cycles

- Speedup: NISC style RISC vs. RISC

DCT	Sort (n element)	Bdist (block 16*h)
2x	1.1x	1.1x

- Speedup: Customized NISC vs. RISC

DCT	Sort (n element)	Bdist (block 16*h)
16x	10x	6.5x



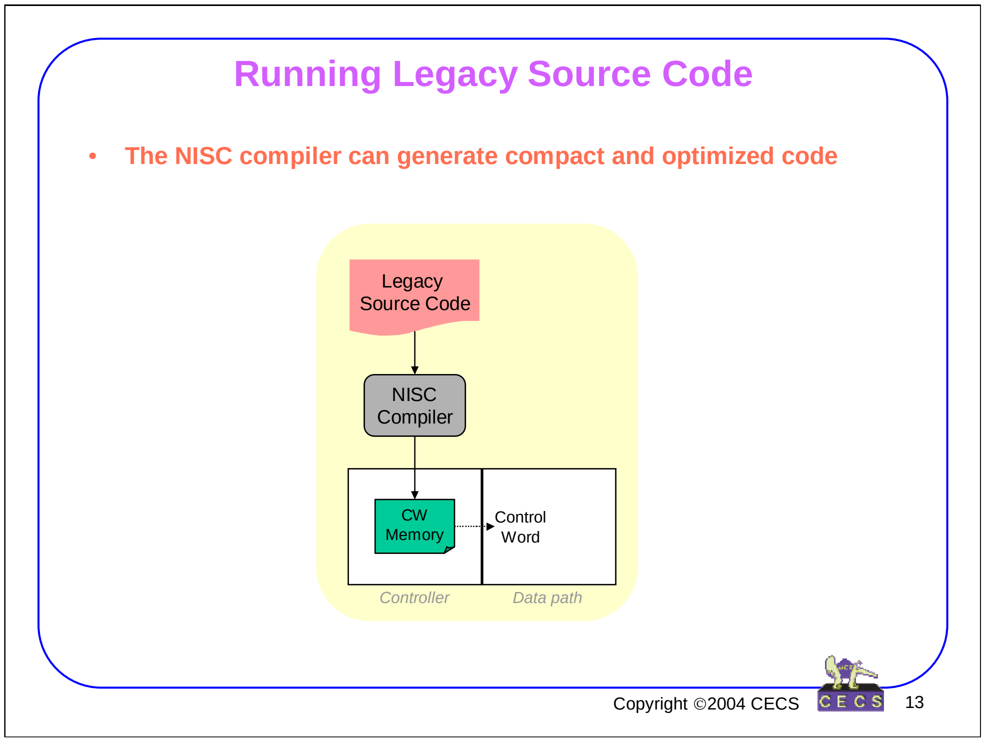
12

NISC vs. RISC Performance

NISC processor with same components and similar connectivity can execute the same code faster since instruction set does not limit parallelism available in the code.

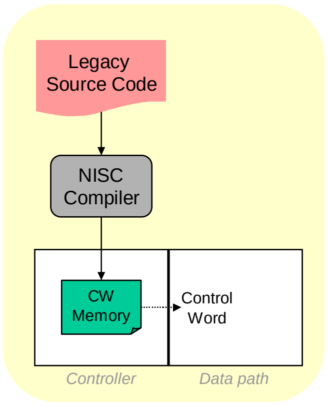
For example, while DCT takes about 8000 cycles to run on RISC, it takes 4000 cycles to run on the same data path controlled by NISC control words.

The performance improvement depends on the compatibility of the data path with the application behavior. For example, on a NISC style RISC with the same components and connectivity as of an instruction-set based RISC, a sort algorithm and the *Bdist* function (a core function in mpeg2 encoder) run only 1.1 times faster. However, on customized data paths with similar components but different connectivity, DCT, sort and *Bdist* run 16, 10 and 6.5 times faster than their RISC versions, respectively.



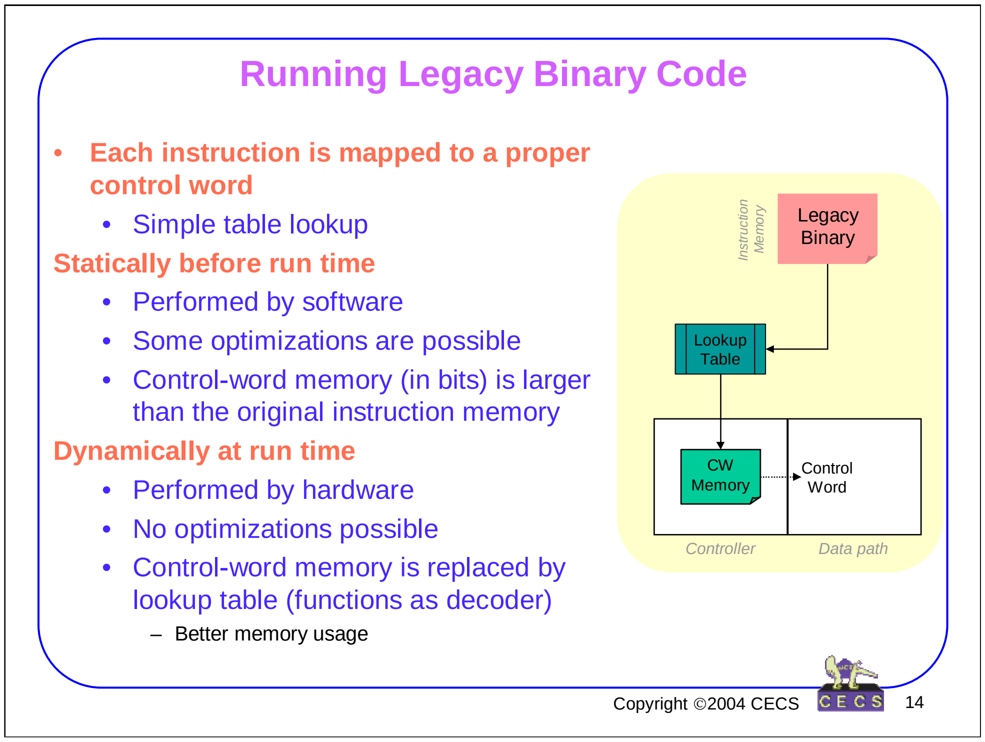
Running Legacy Source Code

- The NISC compiler can generate compact and optimized code



Legacy Code on NISC

If a legacy application is in the form of source code, it can be compiled into compact and optimized control words by the NISC compiler. This control values will be loaded into the control memory of the NISC controller in order to execute the application on the datapath.

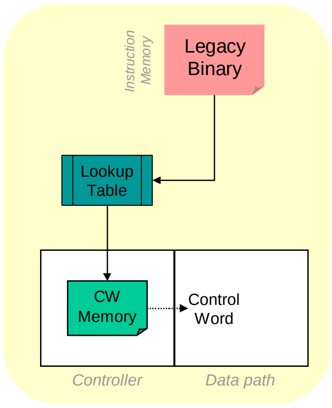


Running Legacy Binary Code

- **Each instruction is mapped to a proper control word**
  - Simple table lookup

Statically before run time
  - Performed by software
  - Some optimizations are possible
  - Control-word memory (in bits) is larger than the original instruction memory

Dynamically at run time
  - Performed by hardware
  - No optimizations possible
  - Control-word memory is replaced by lookup table (functions as decoder)
    – Better memory usage

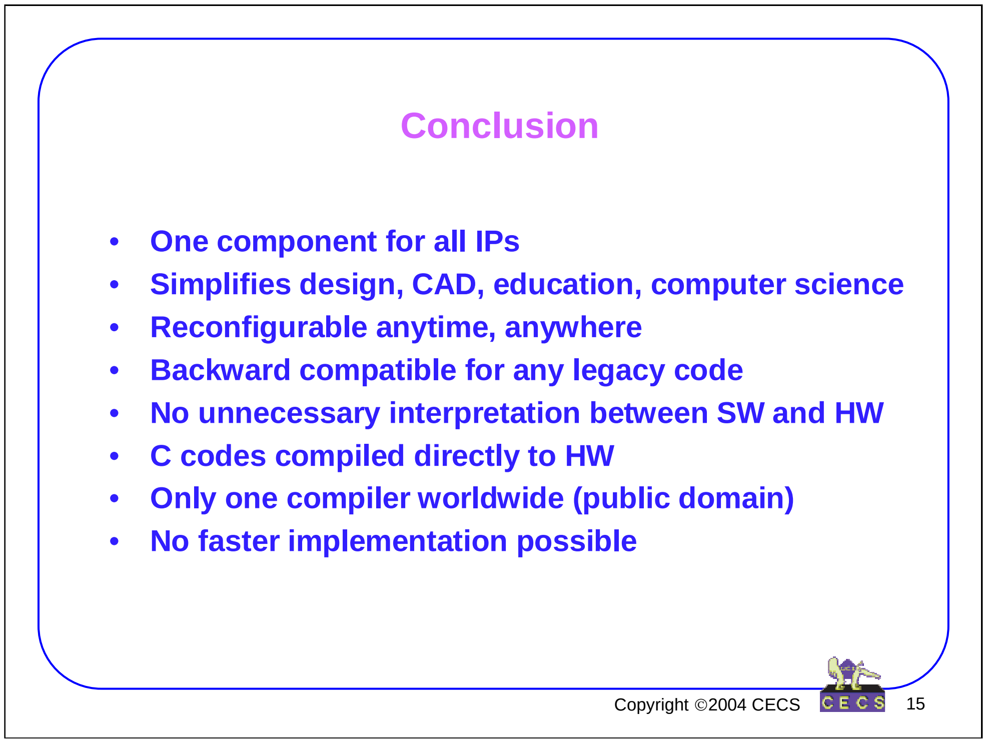


Running Legacy Binary Code and Backward Compatibility

In order to run the legacy binary code on a NISC processor, the binary instructions in the original code must be mapped to proper control words that mimic the behavior of each instruction on the datapath. This translation is basically a table lookup that can be performed statically or dynamically.

The static translation is done by software prior to the execution. Since the software translator has a broader view of the application, it may be able to perform some basic-block optimizations on the code. The result of this translation is finally loaded into the *CW Memory* in the *Controller*. This memory will be larger (in bits) than the original instruction memory that was needed for storing the original binary instructions.

The dynamic translation of binary instructions is performed by hardware. In this case, each instruction is mapped to its corresponding control-word at run time via a look up table. Since each instruction is looked up in the table each time it is executed, the control-word memory in the controller is no longer needed and can be eliminated. The lookup table plays the role of the instruction decoder in standard processors. The performance of the application code in this approach may be less than that of static approach since there is no possibility of local instruction compaction. However, depending on the complexity of the instruction-set of the original architecture (that the legacy binary code was generated for) the total memory usage of this approach can be less than that of static translation, since *CW Memory* is reduced to one register.



Conclusion

- One component for all IPs
- Simplifies design, CAD, education, computer science
- Reconfigurable anytime, anywhere
- Backward compatible for any legacy code
- No unnecessary interpretation between SW and HW
- C codes compiled directly to HW
- Only one compiler worldwide (public domain)
- No faster implementation possible

Conclusion

The NISC processor is the single, necessary and sufficient computational component for design of systems-on-chip (memory is the other necessary and sufficient storage component). NISC is a set of components with different datapaths or controllers and one compiler/simulator.

NISC unifies several concepts from processor architecture, compilers and register-transfer synthesis into one unique concept. Therefore, it simplifies design, education, CAD, testing, IP trade and other aspects of traditional design.

NISC can be reconfigured and reprogrammed statically and dynamically to satisfy power, performance, cost, reliability and other constraints.

Such programmability allows a NISC to emulate other instruction sets.

Since the instruction set is eliminated the C code compiles directly into hardware. There is no unnecessary interpretation between C code and hardware, which allows a NISC to execute any code as fast as semiconductor technology will allow it. In other words, NISC offers the fastest execution of any computer program.