

No-Instruction-Set-Computer (NISC) Technology

Daniel Gajski, Mehrdad Reshadi
Center for Embedded Computer Systems (CECS)
University of California, Irvine
(gajski,reshadi@cecs.uci.edu)



NISC technology benefits

- NISC represents new IP technology
- NISC allows perfect design tuning to application
- NISC converts C to RTL for custom implementation
- NISC enables design for manufacturability

Copyright ©2005 CECS



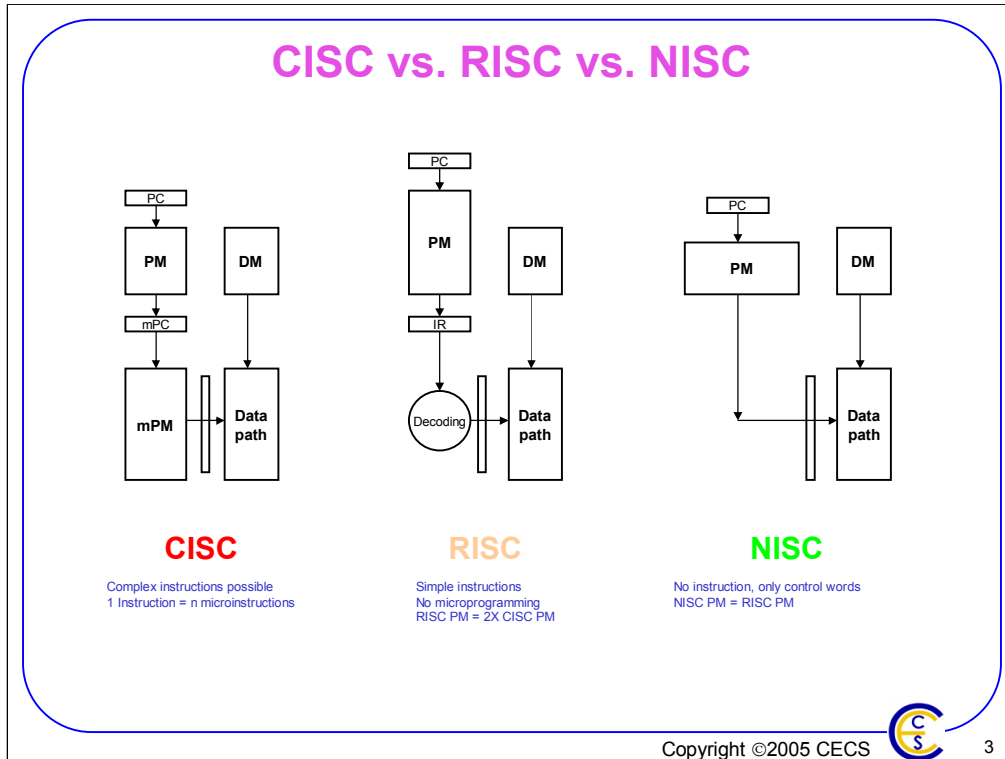
2

NISC Benefits

NISC technology is an enabler for the IP market. It provides a common parametrizable architecture and corresponding compiler and simulator for any IPs that is implemented using NISC technology. The NISC technology allows perfect tuning to any application since the architecture can reflect the structure of the application program while the compiler will optimize executable for such specific architecture.

NISC architecture is defined by a netlist of RTL components such as registers, register files, memories, ALUs, shifters, buses and others. NISC compiler converts C language program into control words in the control memory for the NISC architecture. The content of the control memory together with the netlist can be used as an input to any FPGA, or ASIC standard tools. Since NISC architecture is defined before compilation it can be defined for manufacturability.

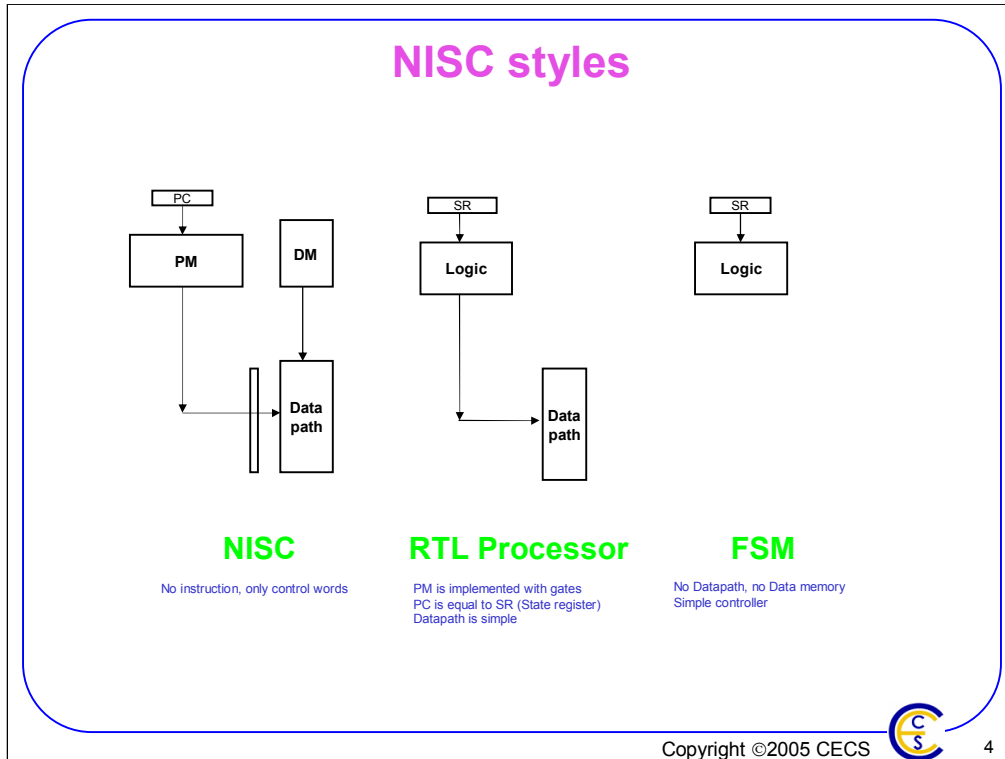
NISC represents a new processor technology since it eliminates the instruction set, the last interpretation step between the programming language and the hardware that executes it.



History of processor architecture

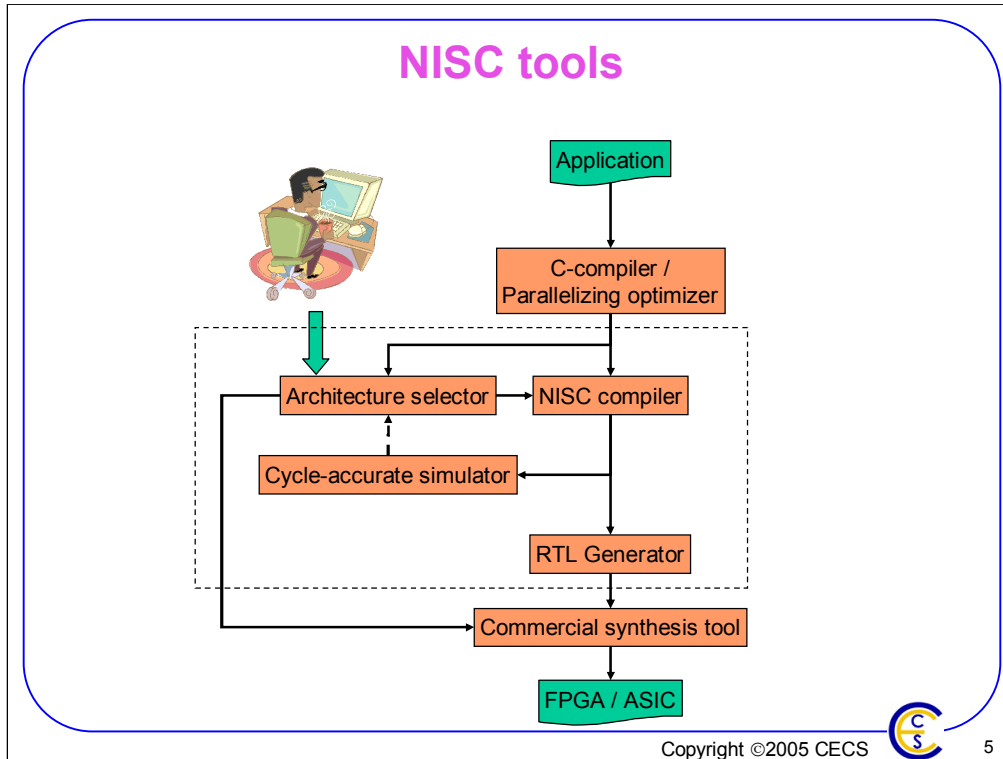
The evolution of the processor architecture can be divided into three phases.

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, while 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 arbitrary instruction sets and creation of specialized instructions, while speeding up the execution.
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 approximately 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) completely removes the decoding 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, which in turn, improves performance substantially.



NISC implementation 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 words in each state then such a NISC is usually called RTL processor. Usually, 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 tool set

NISC tool set consists of several tools:

1. NISC compiler compiles the application C code for the given NISC architecture
2. Architecture selector allows selection of a predefined architecture or capture of a custom architecture specified by the user.
3. NISC Debugger displays the compiled information per clock cycle, pipeline stage and register content.
4. Cycle-accurate simulator simulates the execution of compiled code on given architecture.
5. RTL generator produces Verilog code for downloading to FPGA boards (or ASIC development with standard CAD tools)

NISC methodology

1. Write application in C
2. Select a NISC architecture (manually or automatically)
3. Compile C for the selected architecture
4. Simulate/debug/evaluate the result
5. Repeat 1-4 if not satisfactory
6. Generate RTL for FPGAs/ASICs

Copyright ©2005 CECS

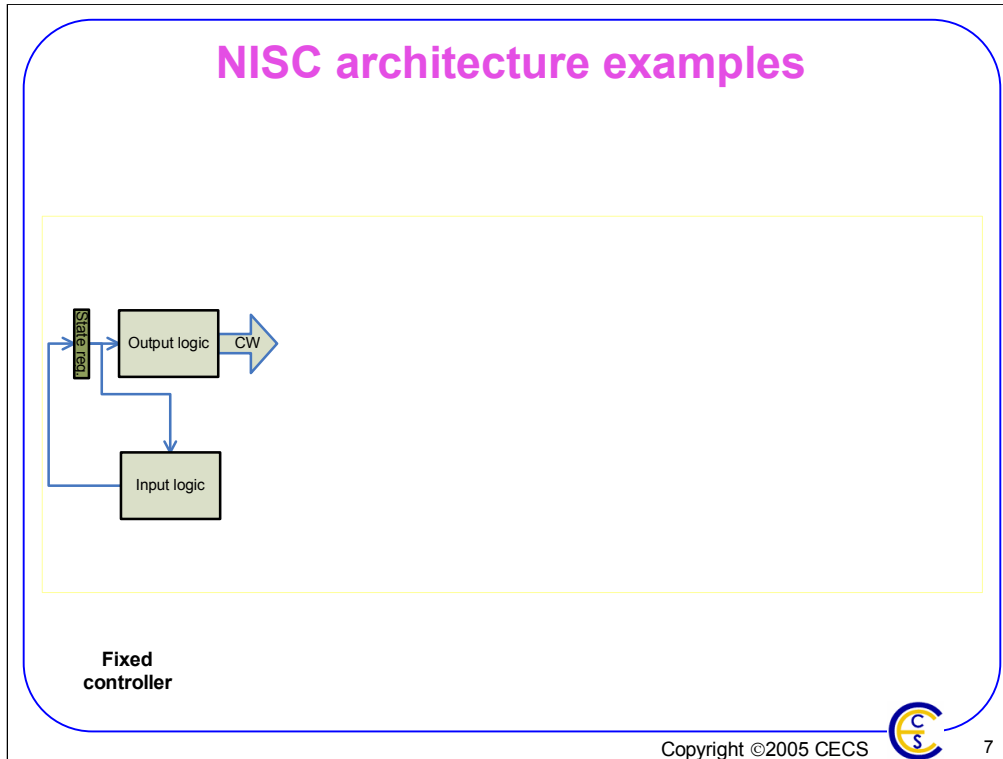


6

NISC Methodology

NISC methodology is very simple and defined to be used by application programmers.

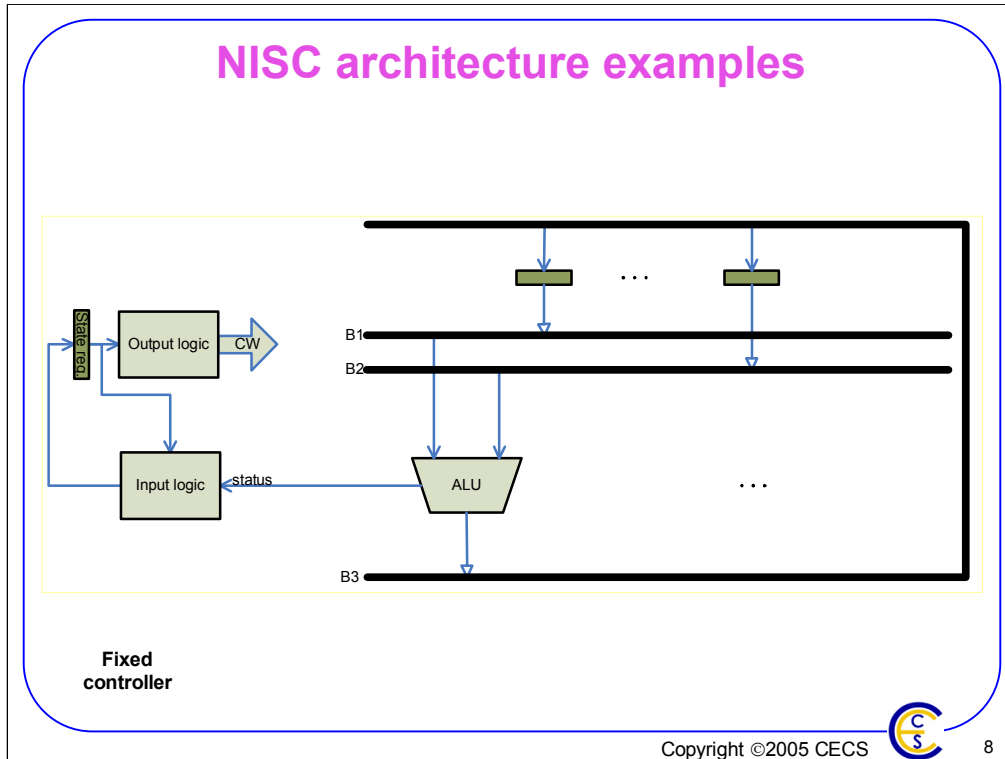
1. A user specifies an application in C programming language.
2. After studying the program structure, the user selects an NISC architecture from the list or defines his/her own architecture by selecting components and their connectivity from a component list.
3. Using the NISC compiler the C program is compiled for the given architecture.
4. After compilation the result can be simulated and evaluated.
5. Improvements can be made by changing original C program or selected architecture.
6. At the end, RTL generator is used to generate Verilog RTL code for FPGA /ASIC implementation.



NISC architecture examples: Fixed Controller

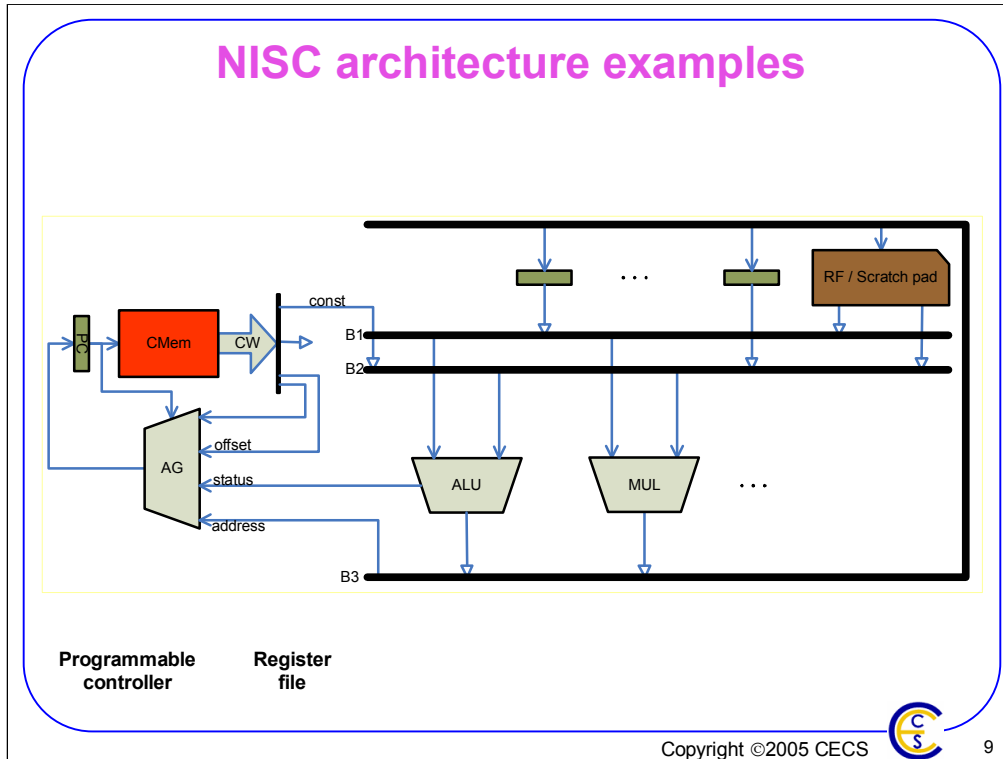
NISC architecture is very general and parametrizable. It is represented by the netlist of RTL components. Many well known architectures are part of NISC technology.

Program Counter can be interpreted as a state register, address generation as the input logic and the Program Memory as the output logic. In this case NISC represents standard FSM, or in other words, a fixed controller whose program is fixed by Output logic



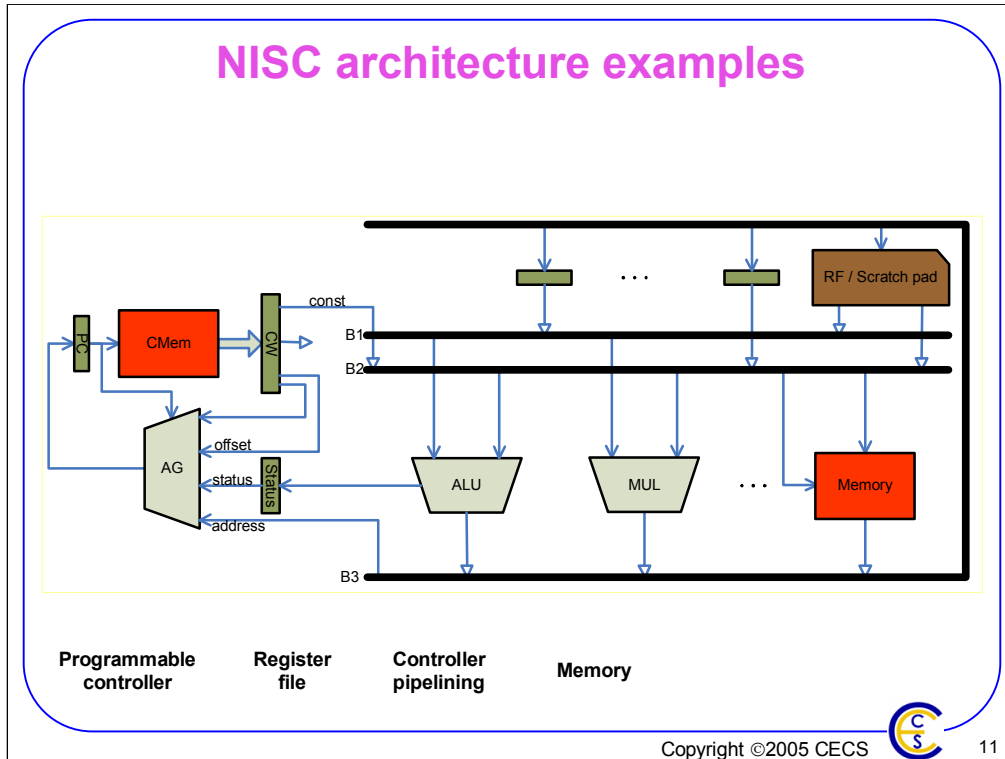
NISC architecture examples: FSM with Datapath (FSMD):

Fixed controller does not perform any computation. Such a controller can be upgraded by adding a simple Datapath consisting of registers, functional units and busses. It is called FSM with Datapath (FSMD). (Equivalent name for FSMD would be a RTL processor which is supported by most of the commercial CAD tools).



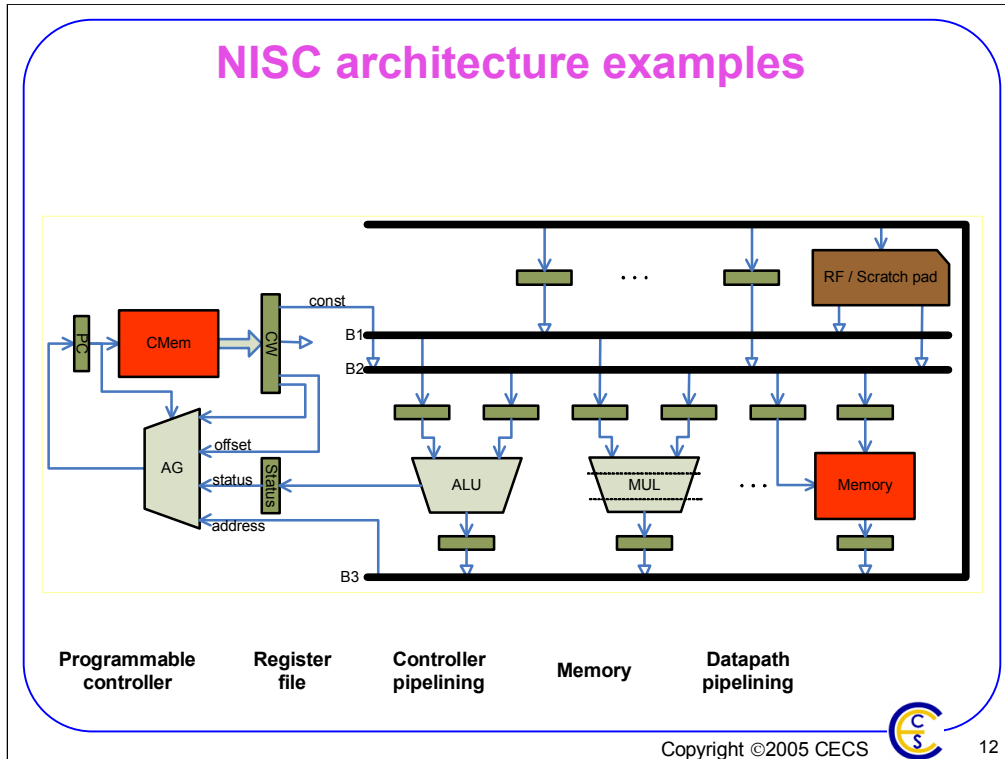
NISC architecture examples: simple processor

The FSMD architecture can be upgraded by replacing Output logic with programmable control word (CW) memory and the input logic with an Address generator (AG). These upgrades make the controller programmable similar to standard processors. The Datapath can be upgraded by adding a Register file (RF) that will make the datapath similar to datapaths of many simple processors.



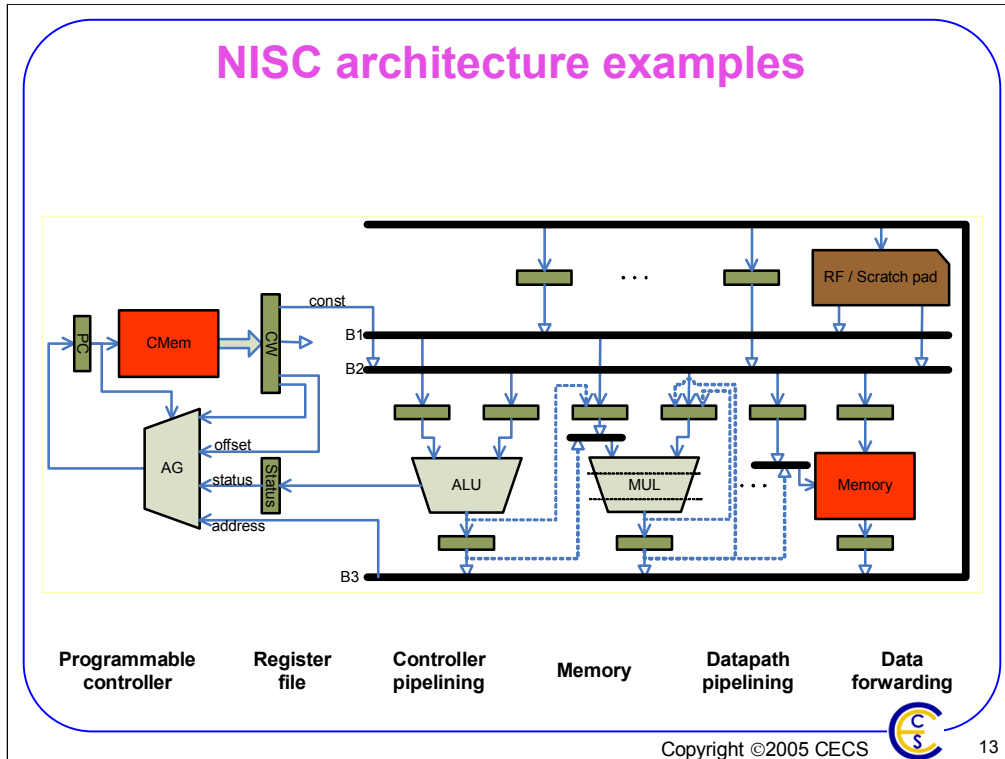
NISC architecture examples: Processor with memory

By adding memory NISC processor can store more data. Such a memory may be sufficient for some application which removes the need for an off-chip memory. Such a memory also introduces a multi-clock access times that increase storing and loading of data. NISC compiler automatically accommodates memory requirements. Similarly, some functional units such as multiplier may need more than one clock cycle to execute. NISC compiler automatically takes into account multi-clock execution during scheduling.



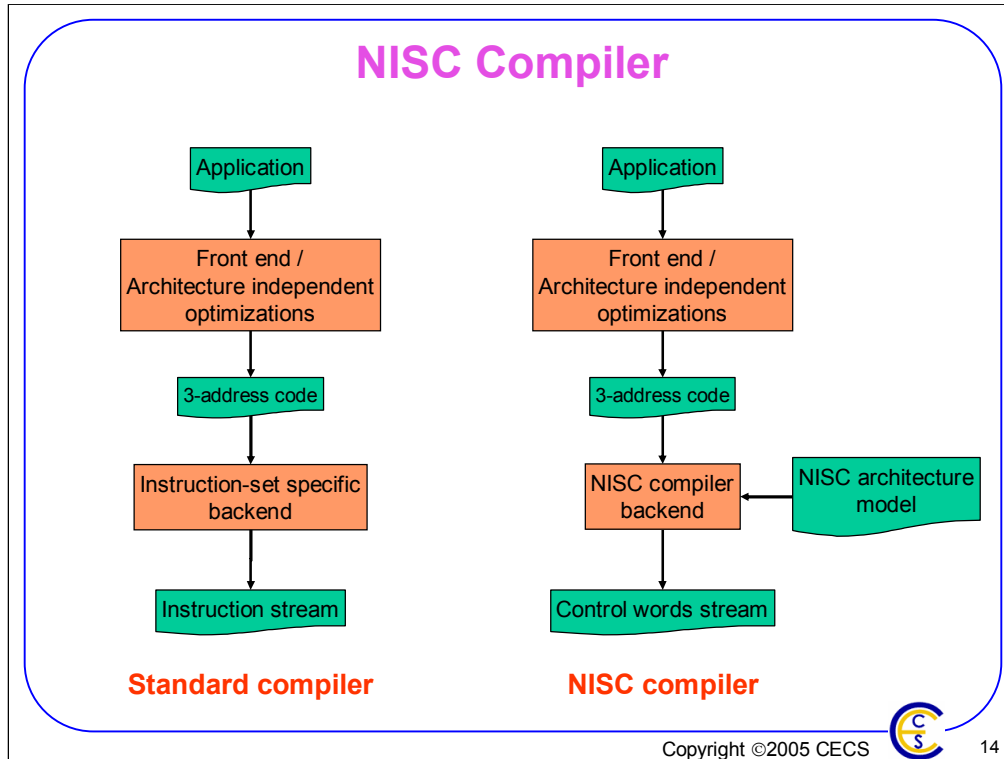
NISC architecture examples: Datapath pipelining

In order to speed up the clock cycle, NISC datapath can be pipelined by adding the input and output registers to functional and storage units and by pipelining some or all of the functional units. However, all units do not need to have the same number of stages. Similarly, all units do not need to have input and output registers. Some units may have only input registers while some others may have only output registers.



NISC architecture examples: Data forwarding

The data in a NISC architecture can be forwarded from one register to another without going through Memory or RF/Scratch pad. This way the computation can be speeded up, although the datapath pipelining becomes non-uniform. However, NISC compiler easily accommodates different length register-to-register pipelining.



NISC Compiler:

NISC compiler is similar to the standard compilers, except for the backend. In a standard compiler each 3-address operation is covered by an instruction from the instruction set. In NISC compiler each 3-address operation is covered by one or more register-to-register transfers in different clock cycles from the given NISC architecture. In a standard compiler only one instruction is executed in each clock cycle. However, in NISC, several register-to-register transfers can also be executed in each clock cycle. The NISC compiler schedules and combines the register-to-register transfers of different 3-address operations in order to achieve maximum performance.

Furthermore, in a standard compiler, if the architecture changes, then (a) someone should define new instructions for the new architecture, (b) the compiler must be updated to incorporate the new instructions. In a NISC compiler, if the architecture changes, the NISC compiler automatically incorporates the changes by analyzing the datapath of the given architecture.

Standard compilers generate instructions. They rely on the instruction decoder in the processor for converting instructions to control words. In instruction-set-based processors, the controller is very complex mostly because of the instruction decoder. Changing the controller, and hence changing the architecture, is very difficult in these processors. NISC does not have any instruction or instruction decoder. Instead, the NISC compiler generates the control words for each clock cycle. As an input to the NISC compiler, the NISC architecture model is represented by a simple controller plus the netlist of datapath components. Therefore, the architectural modifications are much easier in NISC than in traditional processors.

NISC compiler features

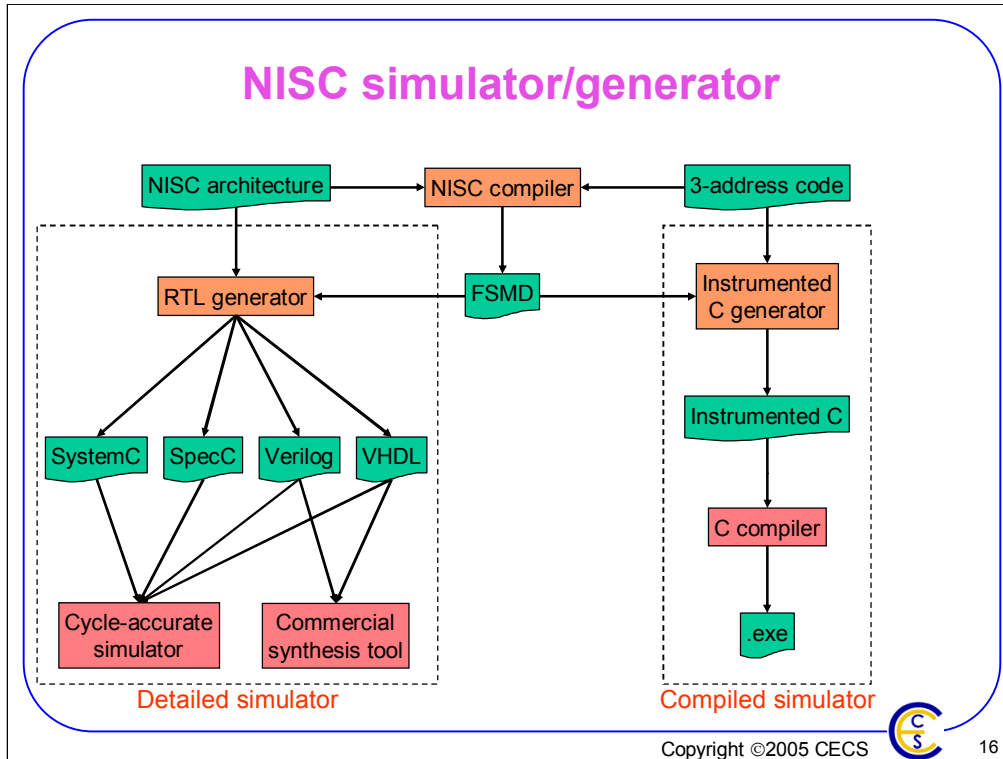
- **When compiling for a NISC architecture NISC compiler supports:**
 - **Operation chaining**
 - Two operations executed sequentially, if possible, in the same clock cycle
 - **Multi-cycle components**
 - Input and output every several cycles
 - **Complex timing diagrams**
 - Memory control
 - **Pipelined functional unit**
 - Input every cycle, output after several cycles
 - **Non-uniform pipelining**
 - Different length data forwarding paths



NISC compiler features

Since NISC compiler maps 3-address code into register-to-register transfers, it tries to execute as many operations as possible in each clock cycle. The compiler reads the clock cycle length and the timings of components from the input model of architecture. Based on these timings, the compiler tries to use the datapath resources in the best possible way. For example, if the combined delay of two operations is less than one clock cycle, then the compiler can schedule both of them serially in one clock cycle. On the other hand, the compiler can use more than one clock cycle for components whose delay is more than one clock cycle. It can also use several clock cycles to accommodate memories with arbitrary access times.

NISC compiler can accommodate many different types of pipelining including functional unit pipelining, datapath pipelining or control pipelining. In datapath pipelining, NISC compiler can accommodate uniform (all paths have the same number of stages) and non-uniform (different paths have different number of stages) pipelining. In control pipelining, the compiler automatically extracts the delay of branches and schedules them properly.



NISC simulation and RTL generation

NISC technology offers two ways of verifying the compilation results:

1. A fast compiled simulator is obtained by converting the results of compilation back to C code. In this C code, the original program is combined (instrumented) with register transfers executed in each clock cycle. In this fast simulation approach, it is also possible to control the level of details that the simulator generates.
2. The other possibility is using commercial tools to simulate the RTL code that is generated after compilation for hardware implementation. In this way, available evaluation and validation tools and methodologies can be used with NISC technology too.

NISC simulator/generator

- **NISC simulator**
 - Converts the 3-address code back to C
 - Combines the NISC information with application code
- **Simulator benefits**
 - Ultimate very fast compiled simulation
 - Used as a profiler it can generate many levels of details
- **NISC generator**
 - Generates RTL code for the NISC design: architecture + binaries
- **Generator benefits**
 - Input to the commercial RTL tools for FPGA/ASIC implementation
 - Very accurate simulation



NISC simulator and generator

To simulate NISC, the 3-address code of the original program is converted back to C and combined with information about its mapping to the NISC architecture. By changing the amount of mapping information that is combined with the 3-address code, the simulator can simulate NISC in many level of details. Additionally, since the program itself is also included in the simulator, it can also be used as a profiler.

NISC generator provides the RTL Verilog code of the final processor. This Verilog code includes both the NISC architecture and the binary contents of program and data memories. The generated RTL code can be used for further implementation with standard commercial CAD tools. It can be also used for very accurate but slower simulation.

NISC debugger

- Displays stages per clock cycle, per pipeline stage
- Displays all register values in each clock cycle
- Displays register transfers in each clock cycle



NISC debugger

NISC debugger helps debug the compiled code, and evaluate the bottlenecks for architecture and code optimization. The debugger displays different view of the status of processor pipeline during execution of program. For each scheduled 3-address operation in the program, the debugger displays the cycle by cycle movement of data values that are used or generated by that operation through all stages or registers of the pipelined architecture. It also displays the schedule of register transfers in different clock cycles and their correspondence to the 3-address operations of the program.

Sample NISC debugger output

clock	operation	0	1	2	3
0(10): \$bb0					
0	t146=(<u>__</u> \$SP+4);	10			
1	t143:*(<u>__</u> \$SP)= <u>__</u> \$LR;	11	4,		
2	t147:*(t146)= <u>__</u> \$FP;	12	<u>__</u> \$SP,4,		
3	no-op	13	t146, <u>__</u> \$SP, <u>__</u> \$LR;		
4	t125: <u>__</u> \$FP= <u>__</u> \$SP;	14	t146, <u>__</u> \$FP,	t146,	
5	t150=(<u>__</u> \$FP+8);	15			
6	t128=(<u>__</u> \$SP+main.stackSize);	16	8,	<u>__</u> \$SP,	

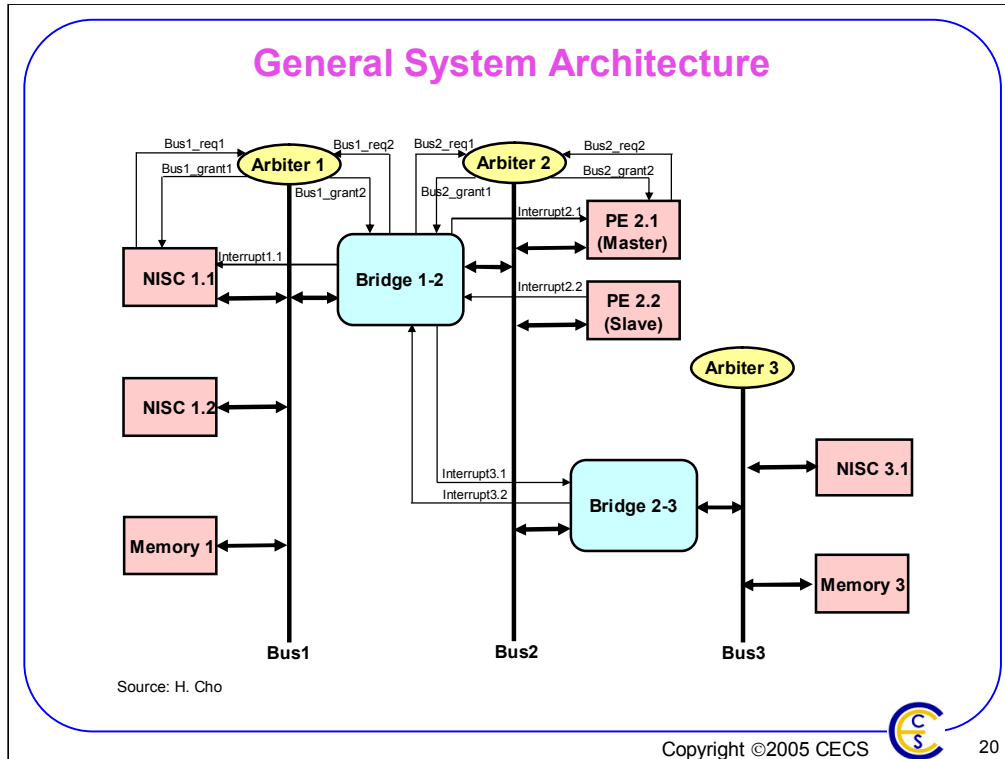
clock	operation	PC	\$0	LR	cwFields	r1	r2	r3	r4	r5	r6	r7	r8	r9	status	RF
0(10): \$bb0																
0	t146=(<u>__</u> \$SP+4);	10														
1	t143:*(<u>__</u> \$SP)= <u>__</u> \$LR;	11			4,											
2	t147:*(t146)= <u>__</u> \$FP;	12				<u>__</u> \$SP,4,										
3	no-op	13						t146,		<u>__</u> \$SP,	<u>__</u> \$LR,					
4	t125: <u>__</u> \$FP= <u>__</u> \$SP;	14								t146,	<u>__</u> \$FP,		t146,			
5	t150=(<u>__</u> \$FP+8);	15														
6	t128=(<u>__</u> \$SP+main.stackSize);	16			8,								<u>__</u> \$SP,			
7	t129: <u>__</u> \$SP= t128; t151:*(t150)= <u>__</u> \$RF(3);	17			main.stackSize,	<u>__</u> \$FP,8,										
8	no-op	18				<u>__</u> \$SP,	main.stackSize,	t150,								
9	no-op	19						t128,		t150,	<u>__</u> \$RF(3),		t150,			
10																t128,
1(20): \$bb1																
0	t115=(<u>__</u> \$FP+4);	20														
1	no-op	21			-4,											
2	t116:*(t115)= 0;	22				<u>__</u> \$FP,-4,										
3	no-op	23			0			t115,								



Debugger output

The above examples show how NISC debugger displays the cycle-by-cycle execution information of 3-address code and the flow of corresponding data values in each pipeline stage or register. The debugger provides this information in several different tables.

A table consists of several sections each corresponding to the basic block of functions of program. Each section shows the clock cycle number, value of the program counter, and the execution of 3-address operations in the pipeline. In the above example, one table shows the flow of data in the pipeline stages while the other table presents the same data flow by showing the data values in different registers for every clock cycle. The color of the data values in the tables is the same as the 3-address operation that uses or produces them.



General System Architecture

NISC technology can be used in design of generic embedded systems by combining different Processing Elements (PEs) with NISC and Memory components. The connection is accomplished through busses which use arbiters for resolution of buss conflicts. If busses have different protocols then the transfer of messages is done through universal bridges. Therefore, any embedded system can be designed using those four (4) components: Processors, NISCs, Bridges and Arbiters. NISC technology offers a path to fast (1-week) prototyping by using this well defined platform architecture and corresponding tools.

Conclusion

- **One component for all computations / communications**
- **Only one compiler/simulator needed for all NISC designs**
- **C codes compiled directly to HW**
- **No faster implementation possible**
- **Unifies SW and HW concepts**
- **Simplifies design, tools, education, design science**
- **Reconfigurable anytime, anywhere**
- **Backward compatible for any legacy code**
- **Any embedded system can be built with set of NISCs**
- **Benefit: Spec-to-Prototype in 1 week**



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). Any computable function defined in programming language C, can be executed on a NISC. NISC is a set of parametrizable templates with different datapaths and/or controllers and one compiler/simulator for all. NISC compiler compiles C language directly into specific NISC architecture specified by a netlist. Since the architecture can be made to reflect the structure of the program, and there is no instruction set, NISC provides the fastest possible implementation for any function.

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 easily. Several NISC can be used to implement almost any embedded system.