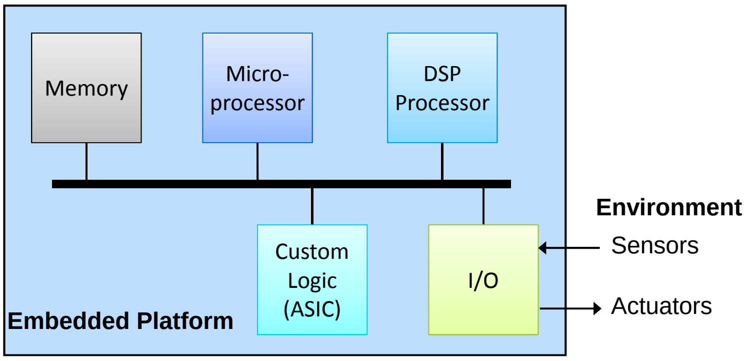
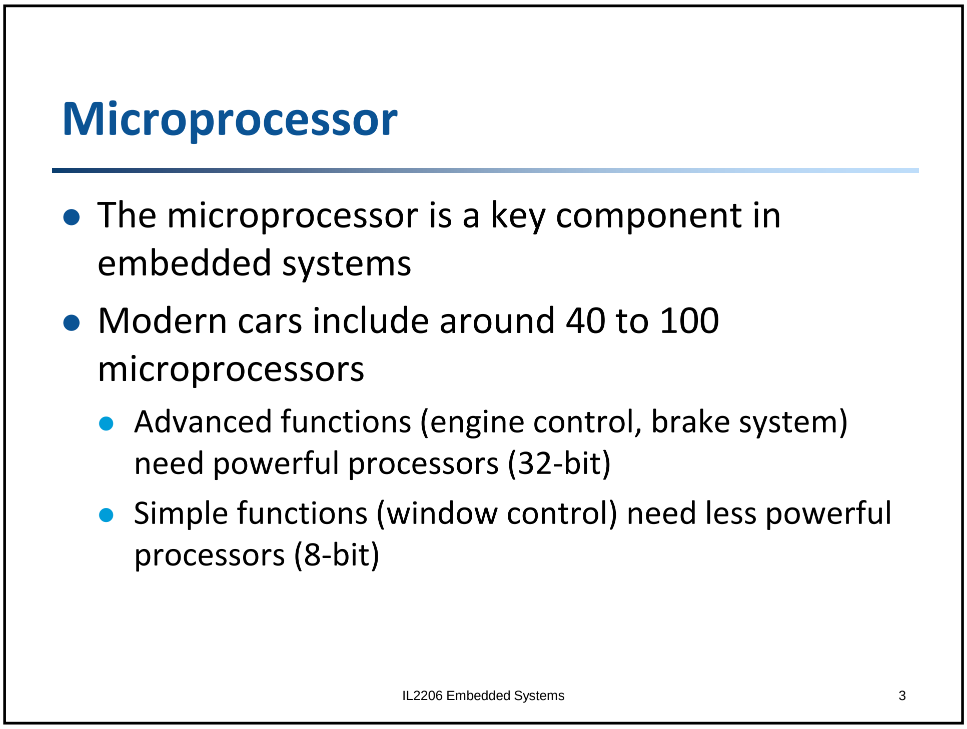
Embedded Computing Platform Microprocessor: Architecture and Instruction Set

Ingo Sander
ingo@kth.se

Microprocessor - A central part of the embedded platform

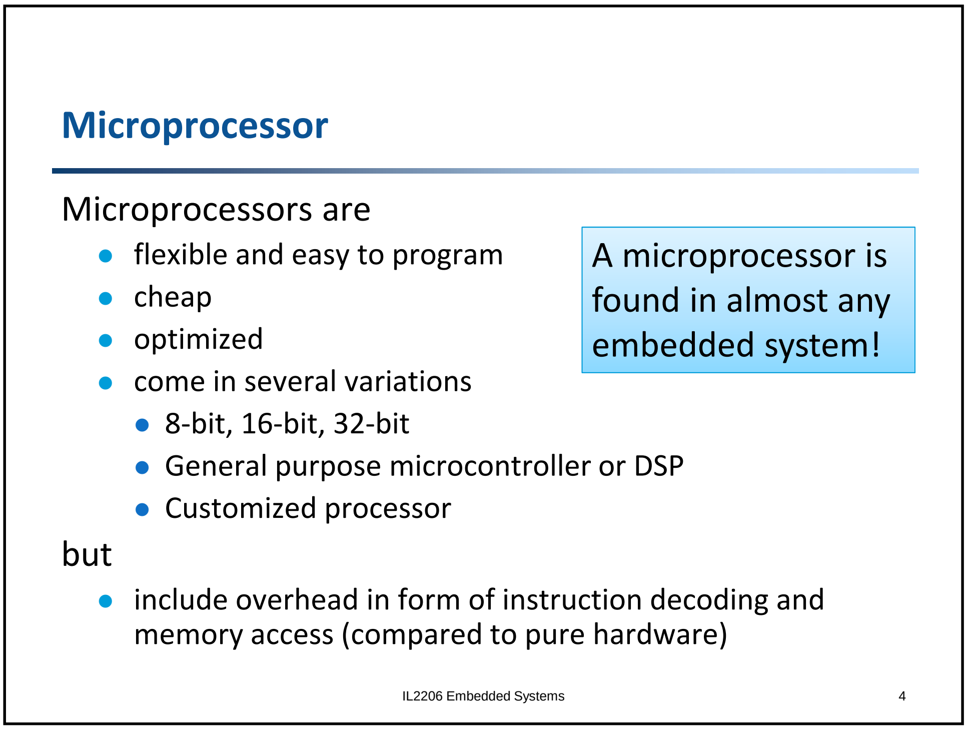
- A platform is the basic *hardware and software architecture* needed for an embedded application





Microprocessor

- The microprocessor is a key component in embedded systems
- Modern cars include around 40 to 100 microprocessors
  - Advanced functions (engine control, brake system) need powerful processors (32-bit)
  - Simple functions (window control) need less powerful processors (8-bit)



Microprocessor

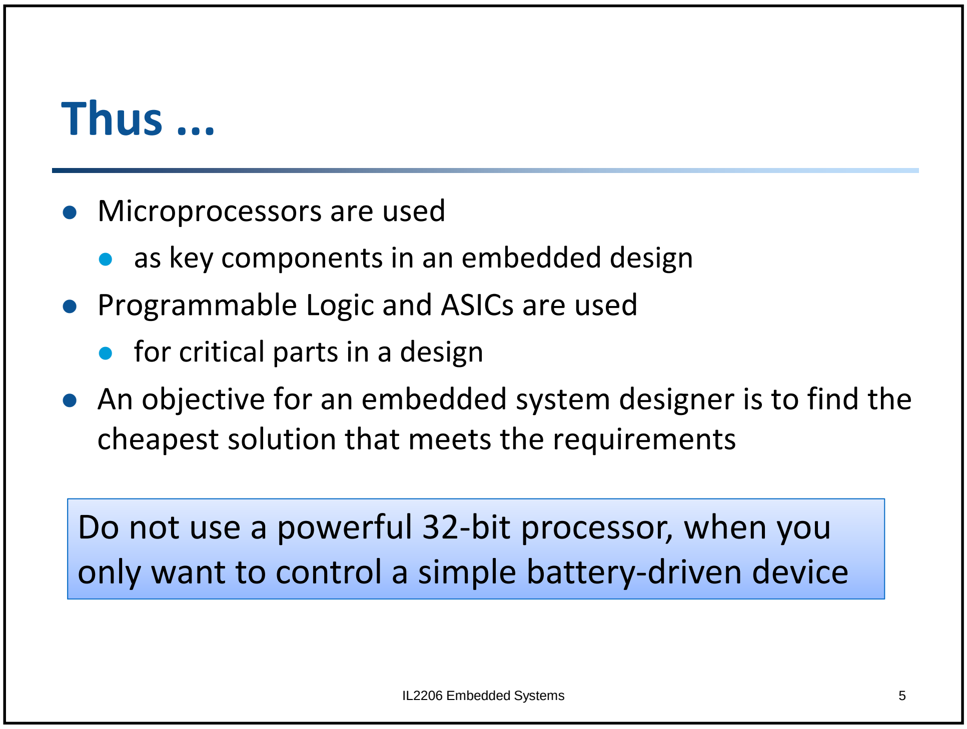
Microprocessors are

- flexible and easy to program
- cheap
- optimized
- come in several variations
  - 8-bit, 16-bit, 32-bit
  - General purpose microcontroller or DSP
  - Customized processor

A microprocessor is found in almost any embedded system!

but

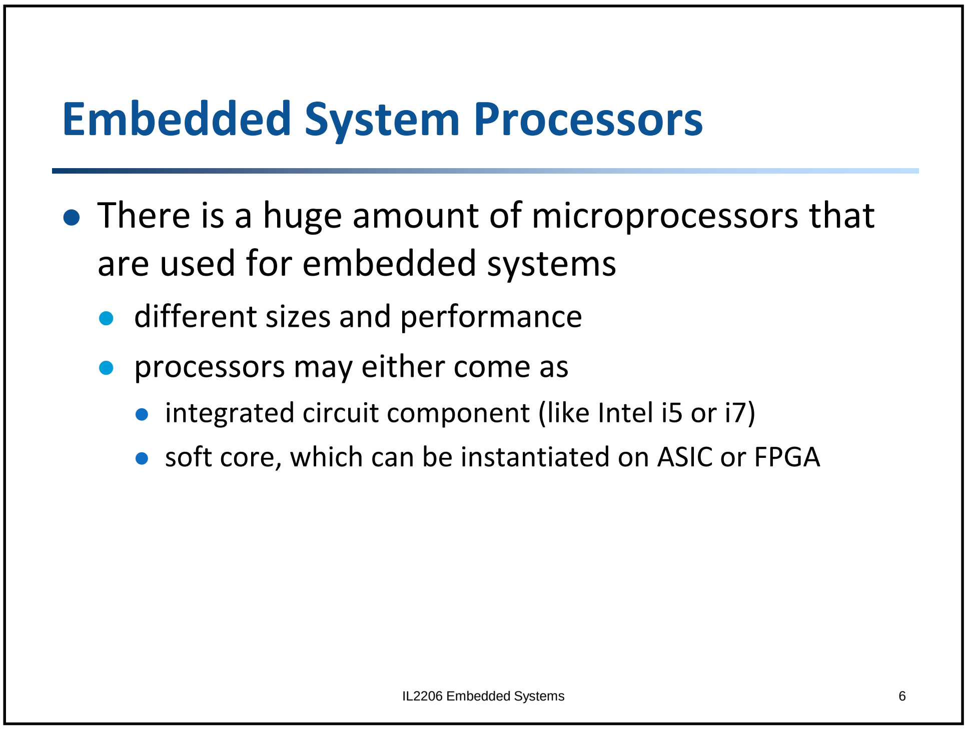
- include overhead in form of instruction decoding and memory access (compared to pure hardware)



Thus …

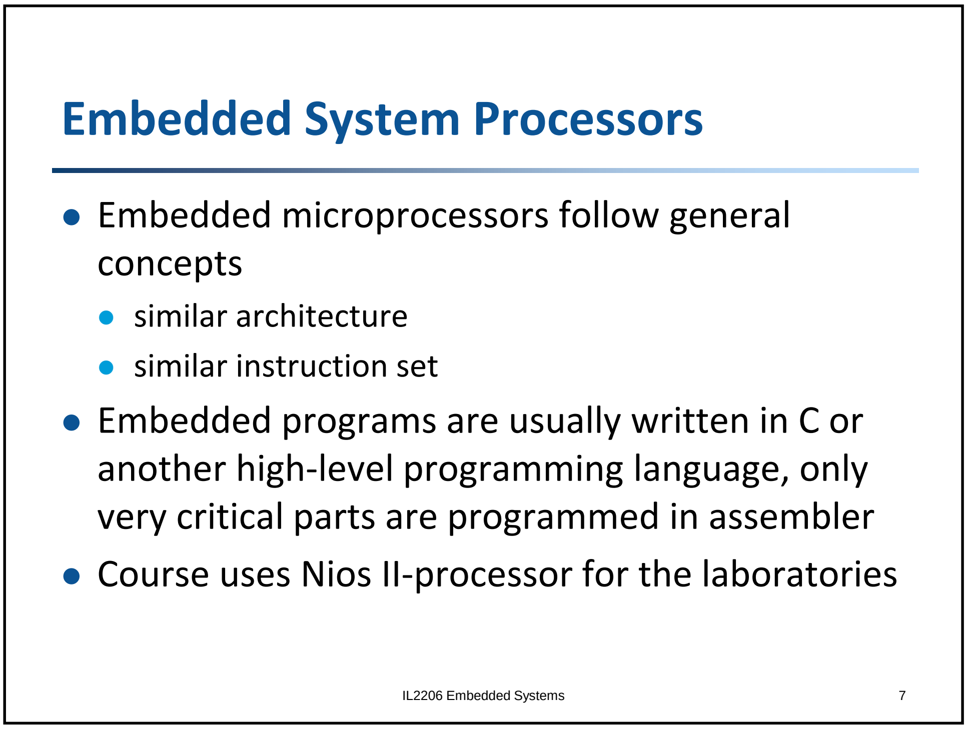
- Microprocessors are used
  - as key components in an embedded design
- Programmable Logic and ASICs are used
  - for critical parts in a design
- An objective for an embedded system designer is to find the cheapest solution that meets the requirements

Do not use a powerful 32-bit processor, when you only want to control a simple battery-driven device



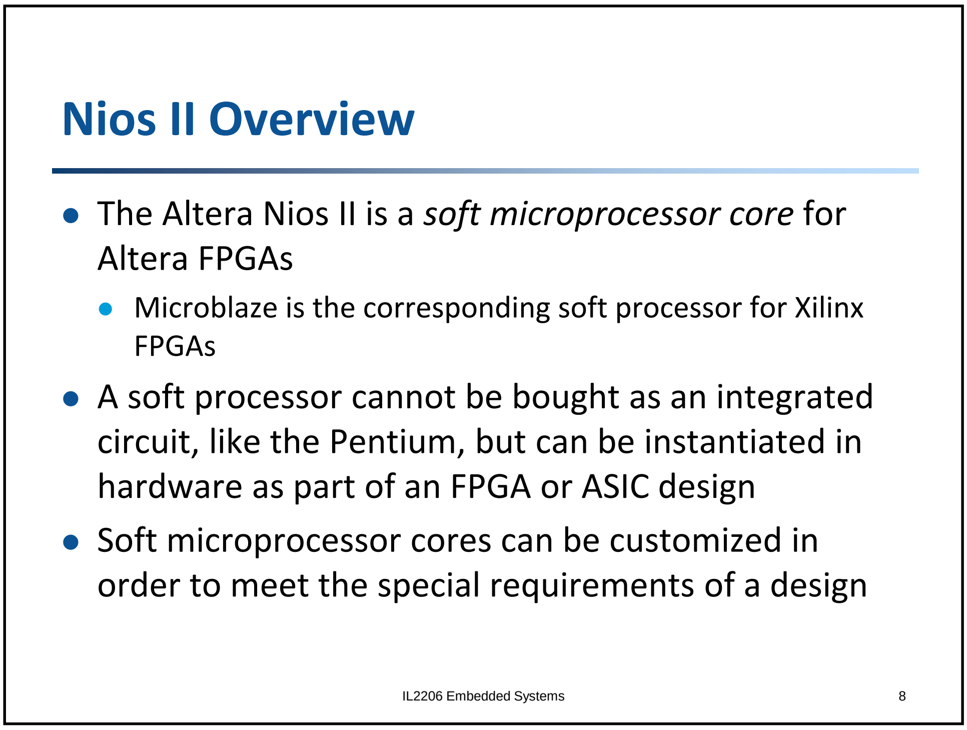
Embedded System Processors

- There is a huge amount of microprocessors that are used for embedded systems
  - different sizes and performance
  - processors may either come as
    - integrated circuit component (like Intel i5 or i7)
    - soft core, which can be instantiated on ASIC or FPGA



Embedded System Processors

- Embedded microprocessors follow general concepts
  - similar architecture
  - similar instruction set
- Embedded programs are usually written in C or another high-level programming language, only very critical parts are programmed in assembler
- Course uses Nios II-processor for the laboratories

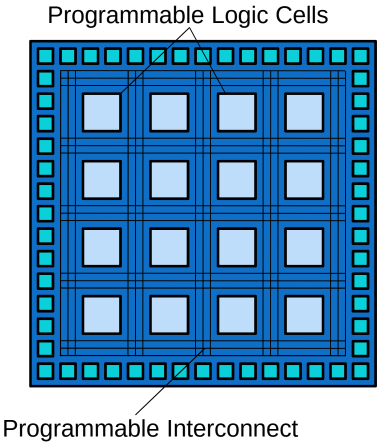


Nios II Overview

- The Altera Nios II is a *soft microprocessor core* for Altera FPGAs
  - Microblaze is the corresponding soft processor for Xilinx FPGAs
- A soft processor cannot be bought as an integrated circuit, like the Pentium, but can be instantiated in hardware as part of an FPGA or ASIC design
- Soft microprocessor cores can be customized in order to meet the special requirements of a design

Field Programmable Gate Arrays

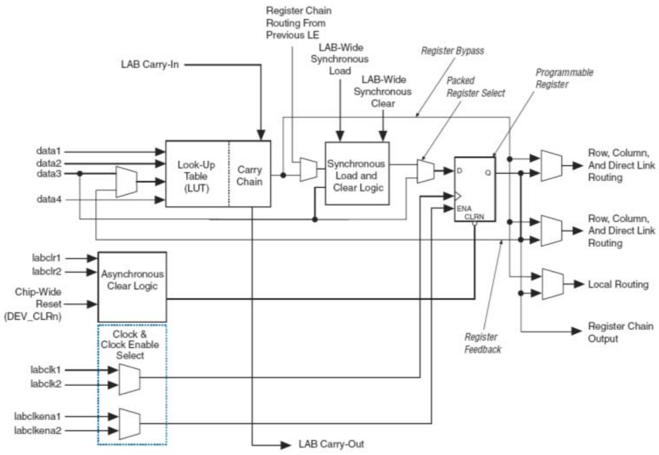
- An FPGA consists of programmable logic cells and a programmable interconnect
- The logic cells are configurable digital hardware blocks
- If large enough, the FPGA can be used to implement any digital functionality

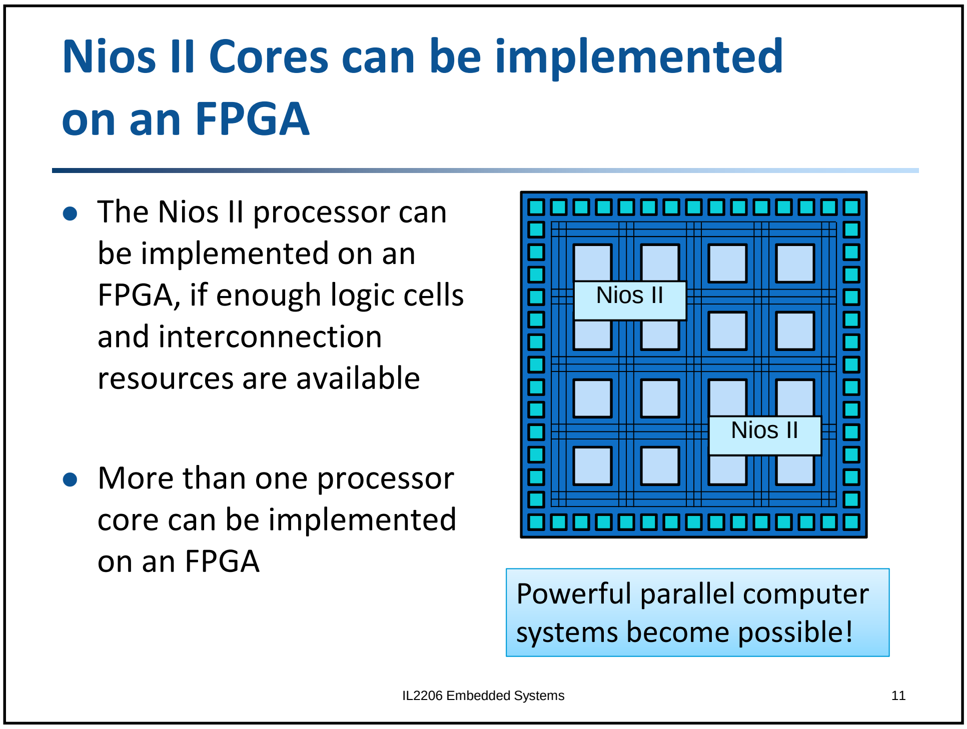


Programmable Logic Cells

Programmable Interconnect

Cyclone II Logic Element

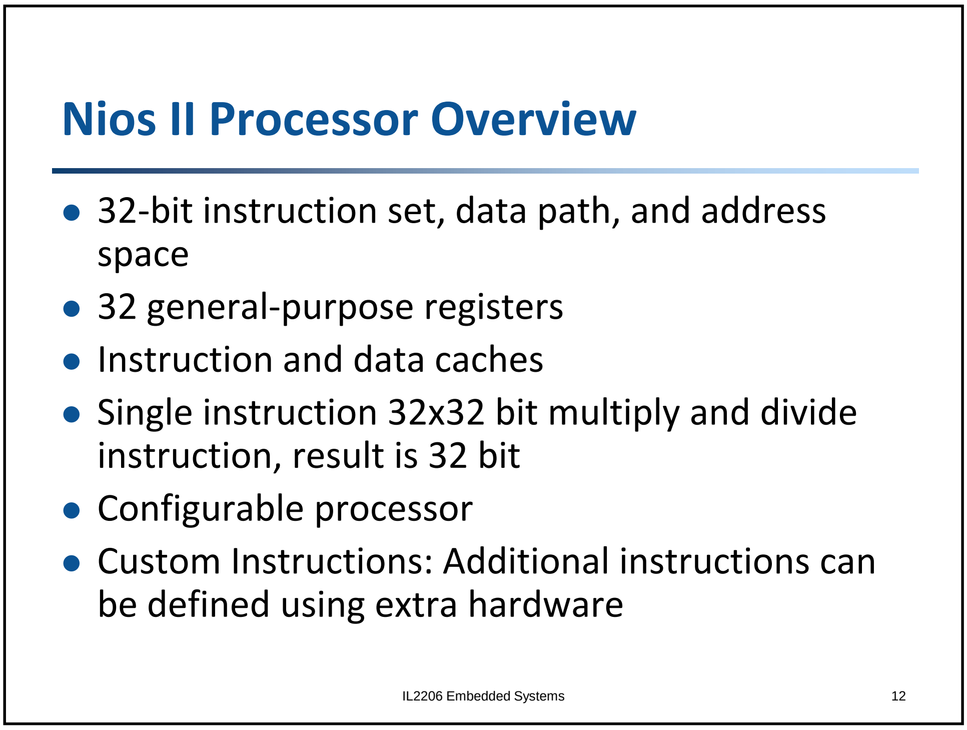



Nios II
Nios II

Nios II Cores can be implemented on an FPGA

- The Nios II processor can be implemented on an FPGA, if enough logic cells and interconnection resources are available
- More than one processor core can be implemented on an FPGA

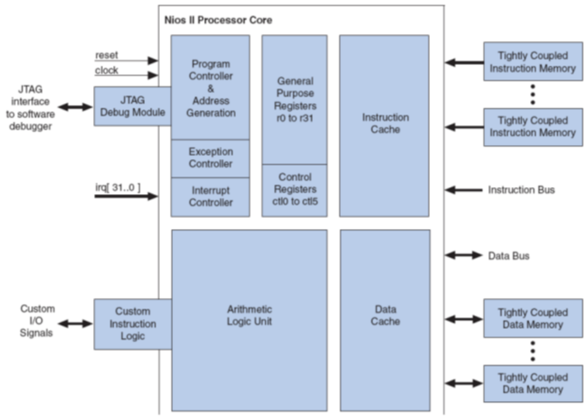
Powerful parallel computer systems become possible!



Nios II Processor Overview

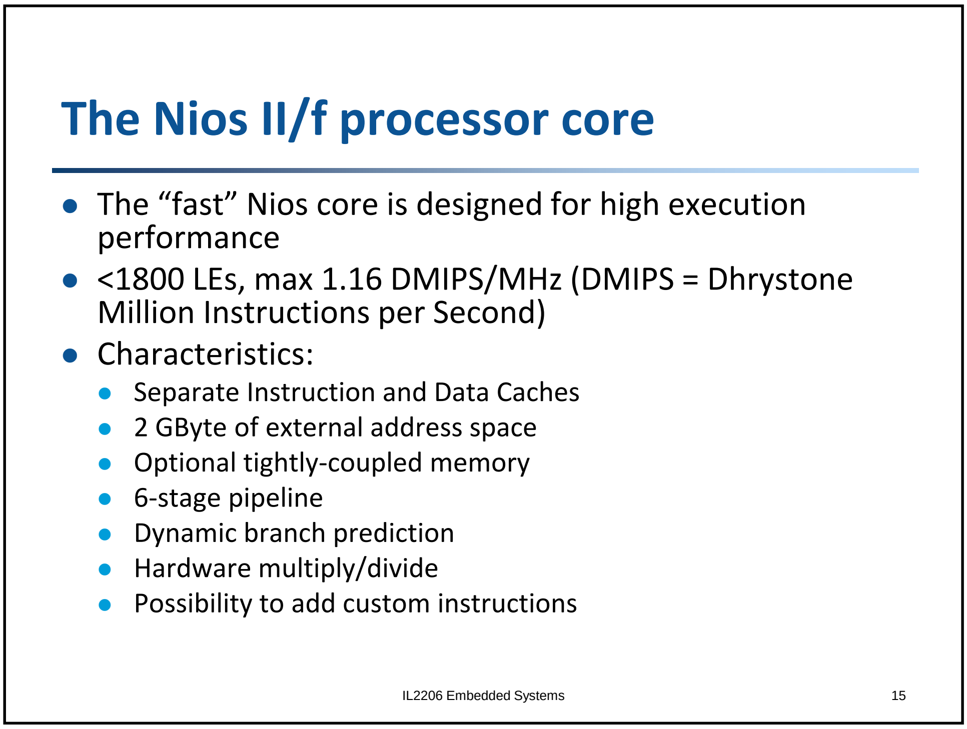
- 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- Instruction and data caches
- Single instruction 32x32 bit multiply and divide instruction, result is 32 bit
- Configurable processor
- Custom Instructions: Additional instructions can be defined using extra hardware

Nios II Block Diagram


Nios II Processor Core
reset
clock
JTAG interface to software debugger
JTAG Debug Module
Program Controller & Address Generation
General Purpose Registers r0 to r31
Instruction Cache
Tightly Coupled Instruction Memory
Tightly Coupled Instruction Memory
Exception Controller
irq[31..0]
Interrupt Controller
Control Registers ctl0 to ctl5
Instruction Bus
Data Bus
Custom I/O Signals
Custom Instruction Logic
Arithmetic Logic Unit
Data Cache
Tightly Coupled Data Memory
Tightly Coupled Data Memory

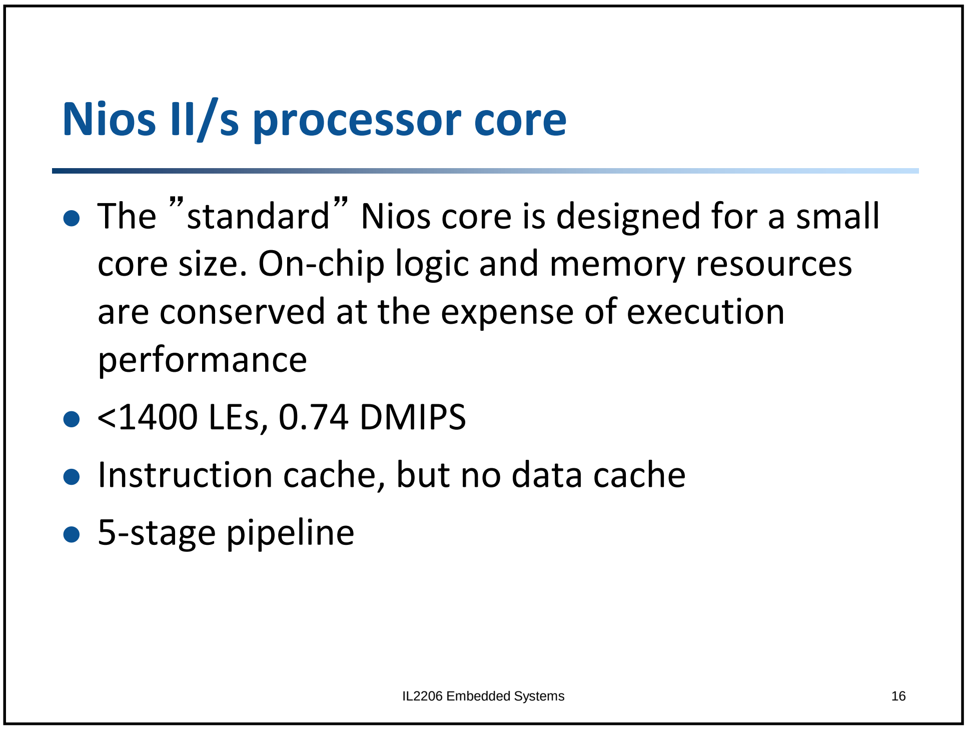
Nios II cores

- When creating a Nios II core the designer can
  - choose a special type of Nios II processor (II/e, II/s, II/f)
  - choose the amount of cache memory and on-chip memory
  - choose peripheral circuits and external memories that will be integrated into the core
  - code is in general portable from one Nios core to another



The Nios II/f processor core

- The “fast” Nios core is designed for high execution performance
- <1800 LEs, max 1.16 DMIPS/MHz (DMIPS = Dhrystone Million Instructions per Second)
- Characteristics:
  - Separate Instruction and Data Caches
  - 2 GByte of external address space
  - Optional tightly-coupled memory
  - 6-stage pipeline
  - Dynamic branch prediction
  - Hardware multiply/divide
  - Possibility to add custom instructions



Nios II/s processor core

- The ”standard” Nios core is designed for a small core size. On-chip logic and memory resources are conserved at the expense of execution performance
- <1400 LEs, 0.74 DMIPS
- Instruction cache, but no data cache
- 5-stage pipeline

Nios II/e processor core

- The ”economy” Nios core is designed to achieve the smallest possible core size
- <700 LEs, 0.15 DMIPS/MHz
- Compatible with Nios II instruction set
- No pipeline
- No caches
- No branch prediction

Overview Nios II family (Extract)

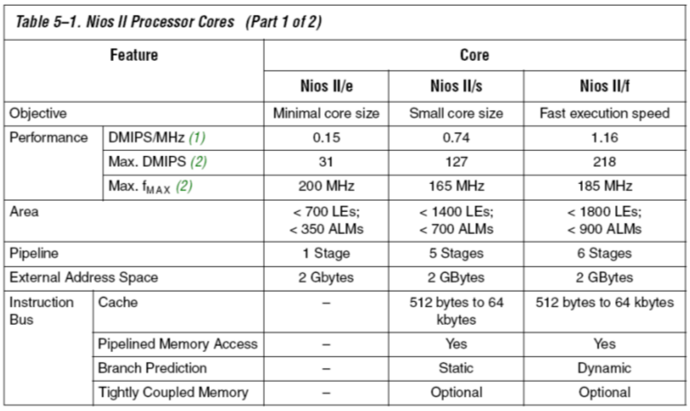
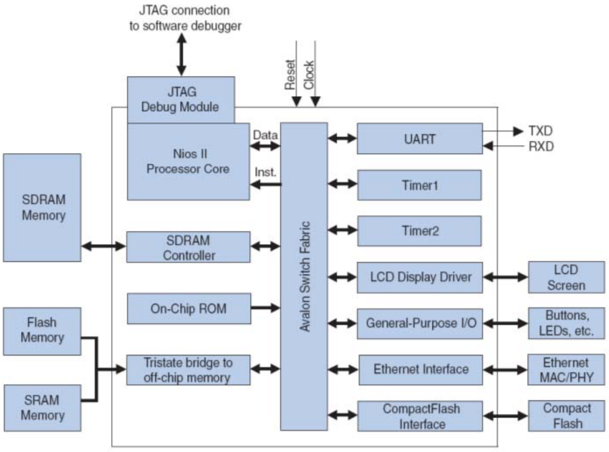


Table 5–1. Nios II Processor Cores (Part 1 of 2)

Feature		Core		
		Nios II/e	Nios II/s	Nios II/f
Objective		Minimal core size	Small core size	Fast execution speed
Performance	DMIPS/MHz *(1)*	0.15	0.74	1.16
	Max. DMIPS *(2)*	31	127	218
	Max. f_{MAX} *(2)*	200 MHz	165 MHz	185 MHz
Area		< 700 LEs; < 350 ALMs	< 1400 LEs; < 700 ALMs	< 1800 LEs; < 900 ALMs
Pipeline		1 Stage	5 Stages	6 Stages
External Address Space		2 Gbytes	2 GBytes	2 GBytes
Instruction Bus	Cache	–	512 bytes to 64 kbytes	512 bytes to 64 kbytes
	Pipelined Memory Access	–	Yes	Yes
	Branch Prediction	–	Static	Dynamic
	Tightly Coupled Memory	–	Optional	Optional

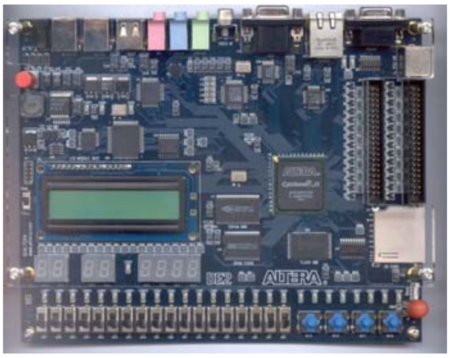
(2) Numbers are for the fastest device in the Stratix II family

Example
Nios II core configuration



The Laboratory Environment

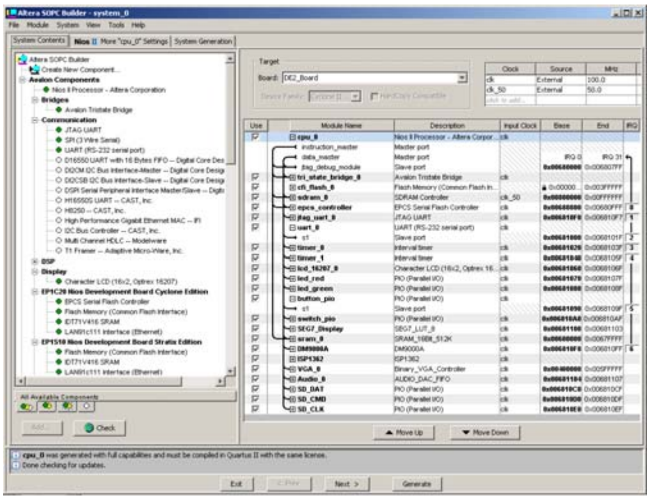
- DE2 Board
  - Cyclone II EP2C35 FPGA
  - 4 Mbytes of flash memory
  - 512 Kbytes of static RAM
  - 8 Mbytes of SDRAM
  - Several I/O-Devices
  - 50 MHz oscillator



Design Flow – Nios II

1. Design a Nios II core with peripherals using the tool SOPC builder (alternatively you can use an existing core)
2. Write your software application using the symbolic addresses and values that have been defined in SOPC builder
3. Download the Nios core to the FPGA
4. Download the software program to the memory

Customizing a Nios II Core
The Tool SOPC Builder


Altera SOPC Builder - system_0
File Module System View Tools Help
System Contents
Nios II More "cpu_0" Settings
System Generation
Altera SOPC Builder
Create New Component...
Avalon Components
Nios II Processor - Altera Corporation
Bridges
Avalon Tristate Bridge
Communication
JTAG UART
SPI (3 Wire Serial)
UART (RS-232 serial port)
D16550 UART with 16 Bytes FIFO -- Digital Core Des
DI2CM I2C Bus Interface-Master -- Digital Core Desig
DI2CSB I2C Bus Interface-Slave -- Digital Core Desig
DSPI Serial Peripheral Interface Master/Slave -- Digit
H16550S UART -- CAST, Inc.
H8250 -- CAST, Inc.
High Performance Gigabit Ethernet MAC -- IFI
I2C Bus Controller -- CAST, Inc.
Multi Channel HDLC -- Modelware
T1 Framer -- Adaptive Micro-Ware, Inc.
DSP
Display
Character LCD (16x2, Optrex 16207)
EP1C20 Nios Development Board Cyclone Edition
EPCS Serial Flash Controller
Flash Memory (Common Flash Interface)
IDT71V416 SRAM
LAN91c111 Interface (Ethernet)
EP1S10 Nios Development Board Stratix Edition
Flash Memory (Common Flash Interface)
IDT71V416 SRAM
LAN91c111 Interface (Ethernet)
All Available Components
Add...
Check
Target
Device Family: Cyclone II
HardCopy Compatible
Move Up
Move Down
Exit
< Prev
Next >
Generate

Board: DE2_Board

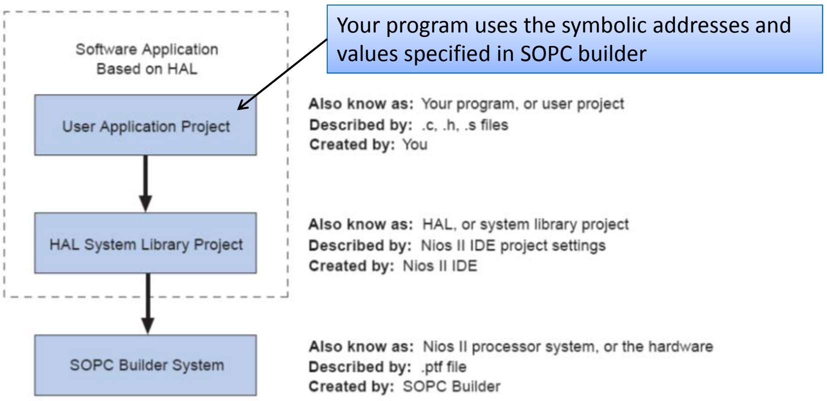
Clock	Source	MHz
clk	External	100.0
clk_50	External	50.0
click to add...		

Use	Module Name	Description	Input Clock	Base	End	IRQ
☑	cpu_0	Nios II Processor - Altera Corpor...	clk			
	instruction_master	Master port				
	data_master	Master port		IRQ 0	IRQ 31	
	jtag_debug_module	Slave port		0x00680000	0x006807FF	
☑	tri_state_bridge_0	Avalon Tristate Bridge	clk			
☑	cfi_flash_0	Flash Memory (Common Flash In...		0x00000...	0x003FFFFF	
☑	sdram_0	SDRAM Controller	clk_50	0x00000000	0x00FFFFFF	
☑	epcs_controller	EPCS Serial Flash Controller	clk	0x00680800	0x00680FFF	0
☑	jtag_uart_0	JTAG UART	clk	0x006810F0	0x006810F7	1
☑	uart_0	UART (RS-232 serial port)	clk			
	s1	Slave port		0x00681000	0x0068101F	2
☑	timer_0	Interval timer	clk	0x00681020	0x0068103F	3
☑	timer_1	Interval timer	clk	0x00681040	0x0068105F	4
☑	lcd_16207_0	Character LCD (16x2, Optrex 16...	clk	0x00681060	0x0068106F	
☑	led_red	PIO (Parallel I/O)	clk	0x00681070	0x0068107F	
☑	led_green	PIO (Parallel I/O)	clk	0x00681080	0x0068108F	
☑	button_pio	PIO (Parallel I/O)	clk			
	s1	Slave port		0x00681090	0x0068109F	5
☑	switch_pio	PIO (Parallel I/O)	clk	0x006810A0	0x006810AF	
☑	SEG7_Display	SEG7_LUT_8	clk	0x00681100	0x00681103	
☑	sram_0	SRAM_16Bit_512K	clk	0x00600000	0x0067FFFF	
☑	DM9000A	DM9000A	clk	0x006810F8	0x006810FF	6
☑	ISP1362	ISP1362	clk			
☑	VGA_0	Binary_VGA_Controller	clk	0x00400000	0x005FFFFF	
☑	Audio_0	AUDIO_DAC_FIFO	clk	0x00681104	0x00681107	
☑	SD_DAT	PIO (Parallel I/O)	clk	0x006810C0	0x006810CF	
☑	SD_CMD	PIO (Parallel I/O)	clk	0x006810D0	0x006810DF	
☑	SD_CLK	PIO (Parallel I/O)	clk	0x006810E0	0x006810EF	

cpu_0 was generated with full capabilities and must be compiled in Quartus II with the same license.
Done checking for updates.

- Designer can select and configure
  - CPU(s)
  - peripherals
- SOPC builder creates core that can be instantiated on FPGA
- Symbolic names are accessible for software designers

The role of the HAL in a Nios II project



Your program uses the symbolic addresses and values specified in SOPC builder

Also know as: Your program, or user project
Described by: .c, .h, .s files
Created by: You

Also know as: HAL, or system library project
Described by: Nios II IDE project settings
Created by: Nios II IDE

Also know as: Nios II processor system, or the hardware
Described by: .ptf file
Created by: SOPC Builder

Reference: Nios II Software Developers Handbook

The system.h file

- The system.h file provides a complete software description of the Nios II system hardware
- The system.h file reflects the actual Nios II hardware, which is given by the *.ptf file.
- A new core means a new *.ptf file and a new system.h file

The system.h file

- The system.h file describes each peripheral and provides the following details:
  - The hardware configuration of the peripheral
  - The base address
  - The IRQ (interrupt request) priority
  - A symbolic name for the peripheral
- If the hardware changes, the source code is still valid, it will only use another system.h file

NEVER edit the system.h file!!!

Quick Question

- Which of the following architectures cannot be realized on the FPGA of the DE2 board (EP2C35)?

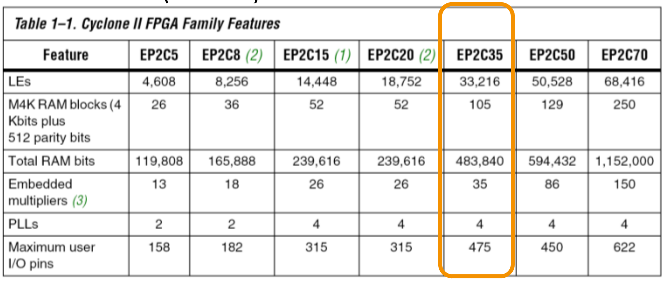


Table 1–1. Cyclone II FPGA Family Features

Feature	EP2C5	EP2C8 (2)	EP2C15 (1)	EP2C20 (2)	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840	594,432	1,152,000
Embedded multipliers (3)	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Maximum user I/O pins	158	182	315	315	475	450	622

- 1) 4 Nios II/f processors (1800LEs) with 4 kB I-Cache and 4 kB D-cache
- 2) 2 Nios II/f processors (1800LEs) with 16 kB I-Cache and 8 kB D-cache
- 3) 1 Nios II/f processor (1800LEs) with 32 kB I-Cache and 32 kB D-cache

Quick Question

- Which of the following architectures cannot be realized on the FPGA of the DE2 board (EP2C35)?

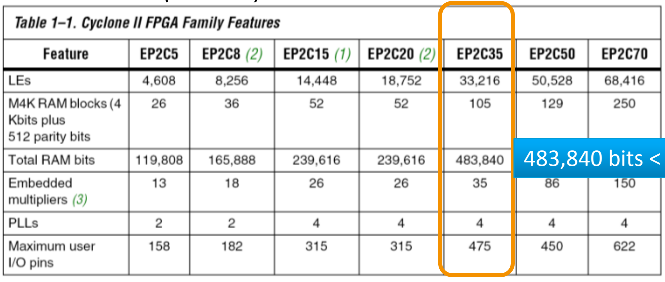


Table 1–1. Cyclone II FPGA Family Features

Feature	EP2C5	EP2C8 (2)	EP2C15 (1)	EP2C20 (2)	EP2C35	EP2C50	EP2C70
LEs	4,608	8,256	14,448	18,752	33,216	50,528	68,416
M4K RAM blocks (4 Kbits plus 512 parity bits	26	36	52	52	105	129	250
Total RAM bits	119,808	165,888	239,616	239,616	483,840		
Embedded multipliers (3)	13	18	26	26	35	86	150
PLLs	2	2	4	4	4	4	4
Maximum user I/O pins	158	182	315	315	475	450	622

483,840 bits < 64 kB

- 1) 4 Nios II/f processors (1800LEs) with 4 kB I-Cache and 4 kB D-cache
- 2) 2 Nios II/f processors (1800LEs) with 16 kB I-Cache and 8 kB D-cache
- 3) 1 Nios II/f processor (1800LEs) with 32 kB I-Cache and 32 kB D-cache

Stratix III Family (announced 2009)

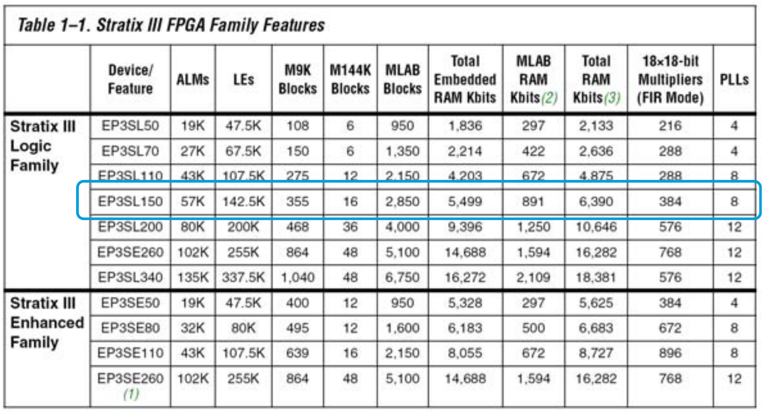


Table 1–1. Stratix III FPGA Family Features

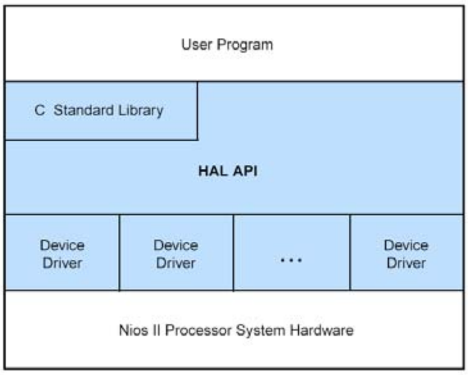
	Device/ Feature	ALMs	LEs	M9K Blocks	M144K Blocks	MLAB Blocks	Total Embedded RAM Kbits	MLAB RAM Kbits (2)	Total RAM Kbits (3)	18×18-bit Multipliers (FIR Mode)	PLLs
Stratix III Logic Family	EP3SL50	19K	47.5K	108	6	950	1,836	297	2,133	216	4
	EP3SL70	27K	67.5K	150	6	1,350	2,214	422	2,636	288	4
	EP3SL110	43K	107.5K	275	12	2,150	4,203	672	4,875	288	8
	EP3SL150	57K	142.5K	355	16	2,850	5,499	891	6,390	384	8
	EP3SL200	80K	200K	468	36	4,000	9,396	1,250	10,646	576	12
	EP3SE260	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12
	EP3SL340	135K	337.5K	1,040	48	6,750	16,272	2,109	18,381	576	12
Stratix III Enhanced Family	EP3SE50	19K	47.5K	400	12	950	5,328	297	5,625	384	4
	EP3SE80	32K	80K	495	12	1,600	6,183	500	6,683	672	8
	EP3SE110	43K	107.5K	639	16	2,150	8,055	672	8,727	896	8
	EP3SE260 (1)	102K	255K	864	48	5,100	14,688	1,594	16,282	768	12

DE3 Board

Todays FPGAs are much larger !!! Check!

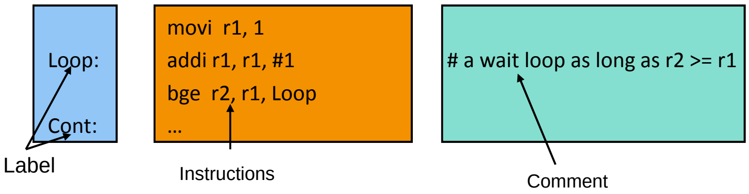
Programming the Nios II

- Altera assumes that the Nios II is programmed in C/C++, using the Hardware Abstraction Layer (HAL)
- Reasons are
  - Maintenance
  - Time-to-Market
  - Flexibility



Nios II Assembly Language

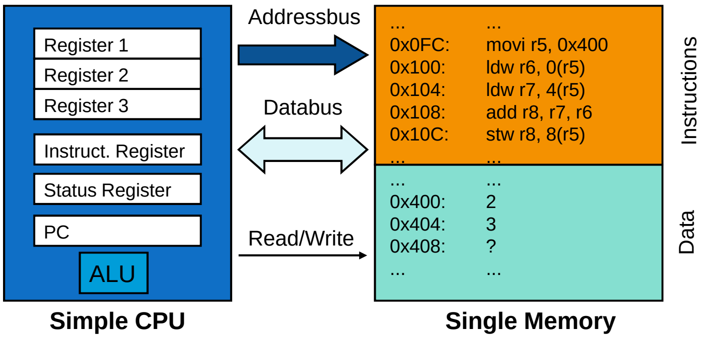
- The assembly language reflects the instruction set (almost one to one)
  - One instruction per line
  - Labels provide names for addresses (usually in first column)
  - Instructions often start in later columns.
  - Columns run to end of line
- Example:



```
Loop:     movi  r1, 1
          addi r1, r1, #1
          bge  r2, r1, Loop        # a wait loop as long as r2 >= r1
Cont:     ...
```

Von Neumann Architecture

- Consists of CPU and one single memory
- Memory holds instructions and data



Register 1

Register 2

Register 3

Instruct. Register

Status Register

PC

ALU

Addressbus

Databus

Read/Write

```
...        ...
0x0FC:     movi r5, 0x400
0x100:     ldw r6, 0(r5)
0x104:     ldw r7, 4(r5)
0x108:     add r8, r7, r6
0x10C:     stw r8, 8(r5)
...        ...
...        ...
0x400:     2
0x404:     3
0x408:     ?
...        ...
```

Instructions

Data

Simple CPU

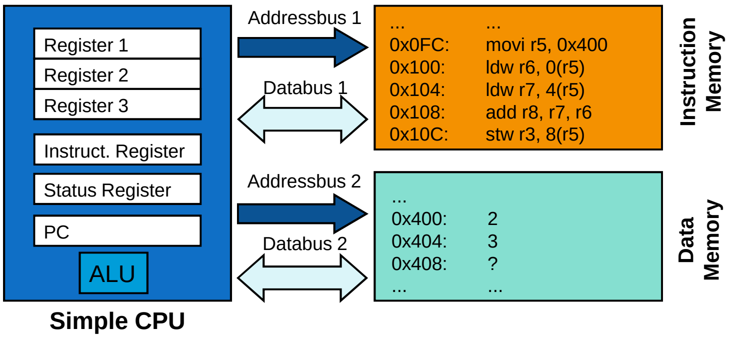
Single Memory

The von Neumann architecture

- Memory holds data, instructions.
- Central processing unit (CPU) fetches instructions from memory.
  - Separate CPU and memory distinguishes programmable computer.
- CPU registers help out: program counter (PC), instruction register (IR), general-purpose registers, etc.

Harvard Architecture

- Consists of CPU and two single memories
- In the original Harvard, one memory holds instructions and the other data

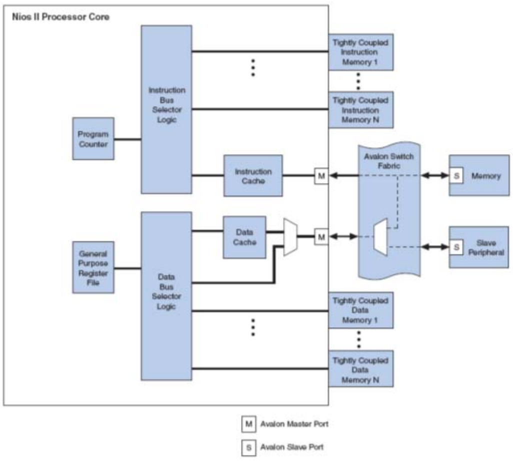


Comparison von Neumann and Harvard

- Harvard allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data:
  - greater memory bandwidth
  - more predictable bandwidth
- Additional hardware, since two address and data busses are needed

Nios II Memory & I/O Organization

- Harvard Architecture (Separated Instruction & Data Memory)
- Each peripheral circuits is assigned an individual address range (memory-mapped I/O)


Nios II Processor Core
Instruction Bus Selector Logic
Program Counter
Instruction Cache
Data Cache
General Purpose Register File
Data Bus Selector Logic
Tightly Coupled Instruction Memory 1
Tightly Coupled Instruction Memory N
Avalon Switch Fabric
M
M
S
Memory
S
Slave Peripheral
Tightly Coupled Data Memory 1
Tightly Coupled Data Memory N
M Avalon Master Port
S Avalon Slave Port

Addresses and Endianness

- The Nios II uses 32-bit addresses
- A Word is 32 bits (4 bytes) long
- An Address refers to a byte (not a word)
- The Nios II processor uses the little-endian memory system
  - Little-endian: lowest-order byte resides in the low-order bits of a word
  - Big-endian: lower-order byte resides in highest bits of the word

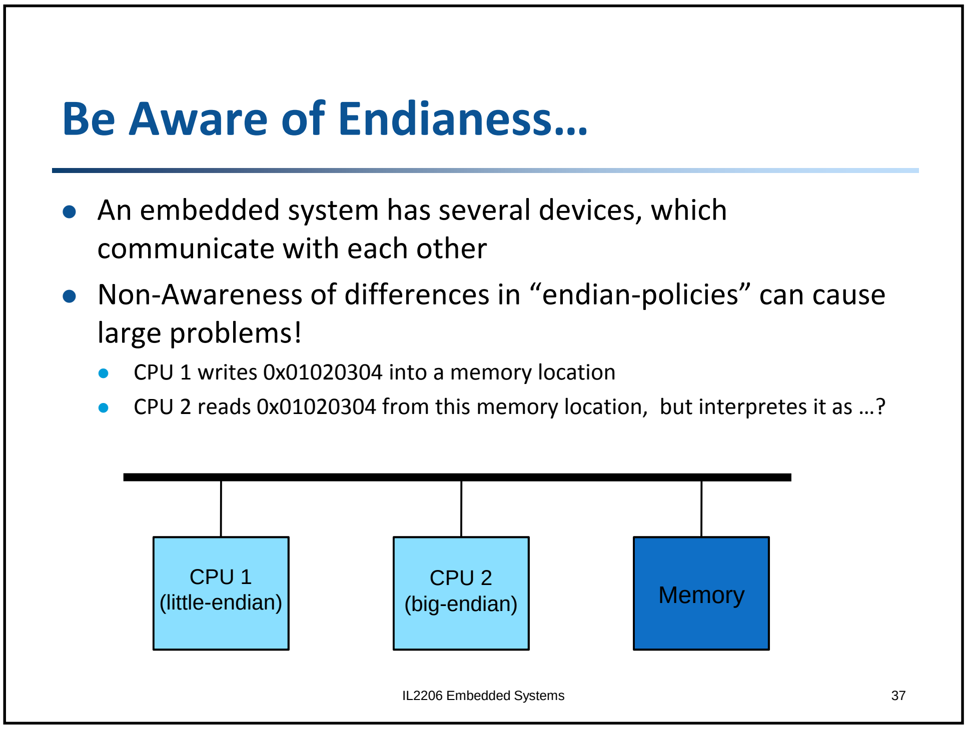
	bit 31			bit 0
Register:	0x0A	0x0B	0x0C	0x0D

Memory (big endian):

x:	0x0A
x+1:	0x0B
x+2:	0x0C
x+3:	0x0D

Memory (little endian):

x:	0x0D
x+1:	0x0C
x+2:	0x0B
x+3:	0x0A



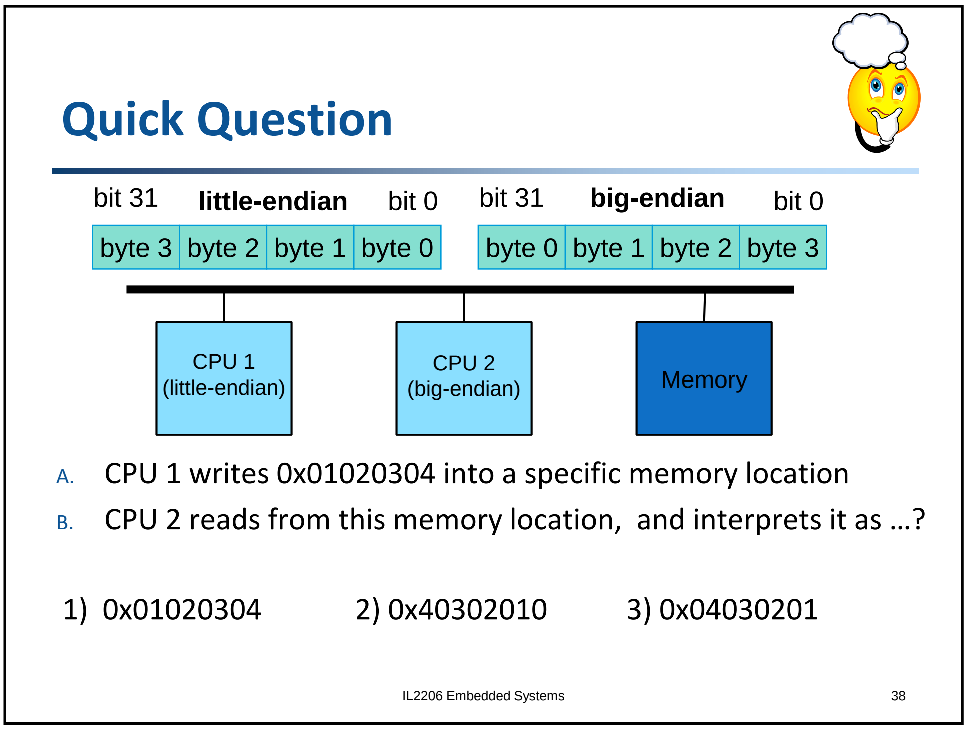
Be Aware of Endianess…

- An embedded system has several devices, which communicate with each other
- Non-Awareness of differences in “endian-policies” can cause large problems!
  - CPU 1 writes 0x01020304 into a memory location
  - CPU 2 reads 0x01020304 from this memory location, but interpretes it as …?

CPU 1 (little-endian)

CPU 2 (big-endian)

Memory



Quick Question

bit 31 **little-endian** bit 0

byte 3 | byte 2 | byte 1 | byte 0

bit 31 **big-endian** bit 0

byte 0 | byte 1 | byte 2 | byte 3

CPU 1 (little-endian)

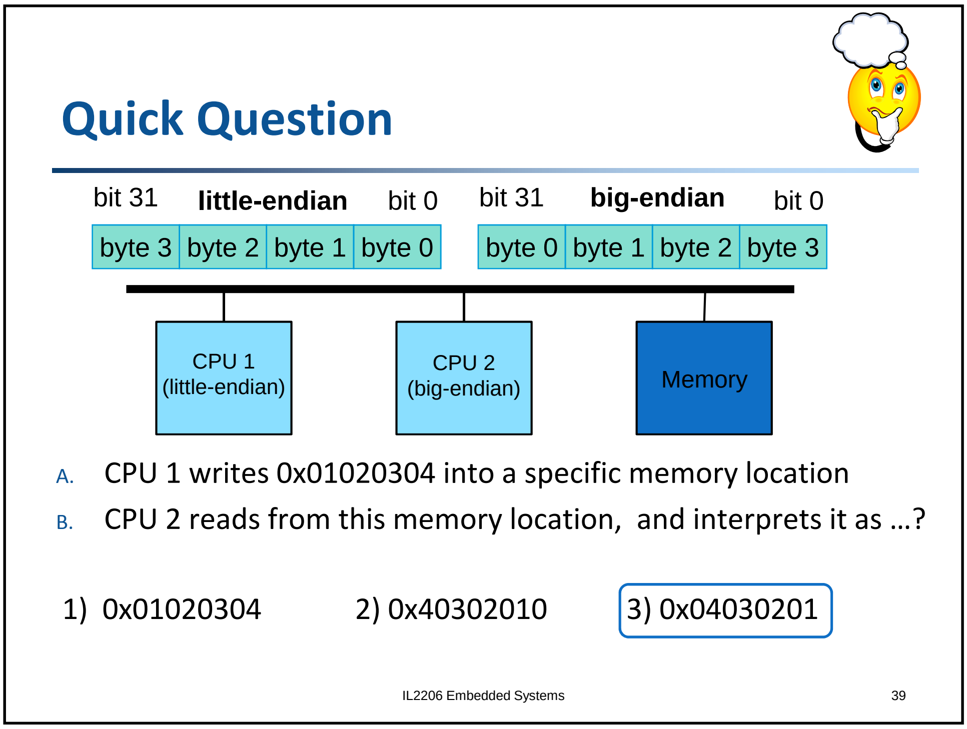
CPU 2 (big-endian)

Memory

A. CPU 1 writes 0x01020304 into a specific memory location

B. CPU 2 reads from this memory location, and interprets it as …?

1) 0x01020304 2) 0x40302010 3) 0x04030201



Quick Question

bit 31 **little-endian** bit 0

byte 3	byte 2	byte 1	byte 0

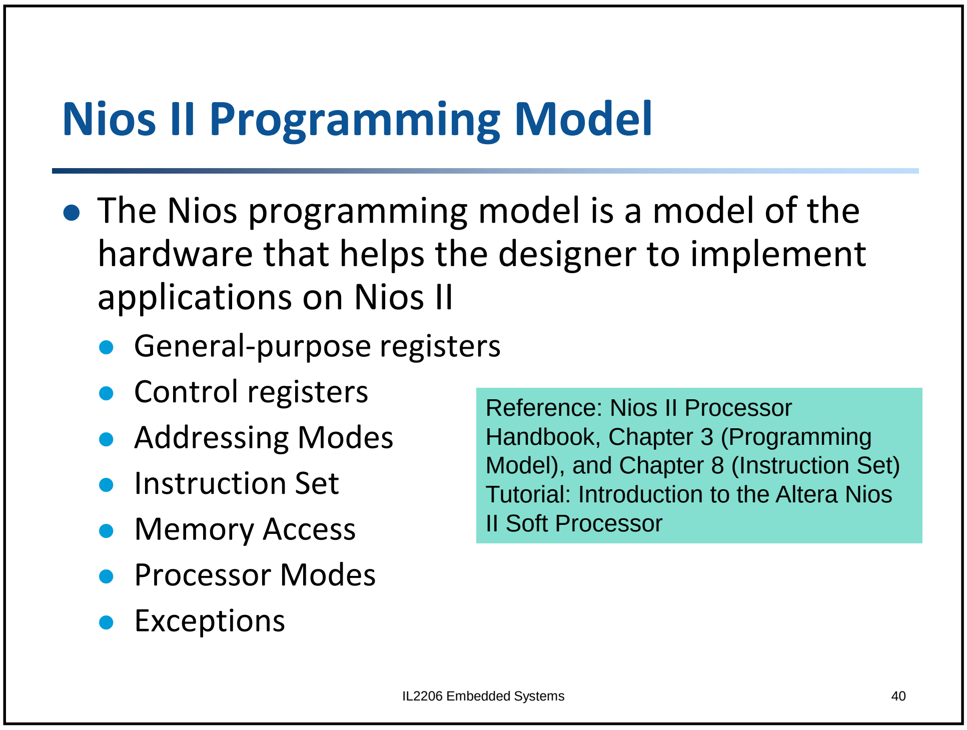
bit 31 **big-endian** bit 0

byte 0	byte 1	byte 2	byte 3

A. CPU 1 writes 0x01020304 into a specific memory location

B. CPU 2 reads from this memory location, and interprets it as …?

1) 0x01020304 2) 0x40302010 3) 0x04030201



Nios II Programming Model

- The Nios programming model is a model of the hardware that helps the designer to implement applications on Nios II
  - General-purpose registers
  - Control registers
  - Addressing Modes
  - Instruction Set
  - Memory Access
  - Processor Modes
  - Exceptions

Reference: Nios II Processor Handbook, Chapter 3 (Programming Model), and Chapter 8 (Instruction Set)
Tutorial: Introduction to the Altera Nios II Soft Processor

Nios II Programming Model

- **Students shall study the programming model on their own**
  - only few slides on the programming model are presented in the lecture and are left to self-studies
- Please read the tutorial “Introduction to the Altera Nios II Soft Processor” (available via the course web page)

Nios II Programming Model Addressing Modes

- The Nios II architecture supports the following addressing modes:
  - *Register addressing*: all operands are registers, e.g. `add r6, r7, r8`
  - *Displacement addressing*: address is calculated as the sum of a register and a signed, 16-bit immediate value, e.g. `ldw r6, 100(r5)`
  - *Immediate addressing*: the operand is a constant within the instruction itself, e.g. `movi r6, -30`
  - *Register indirect addressing*: as in displacement adressing, but the displacement is the constant 0, e.g. `ldw r6, (r5)`
  - *Absolute addressing*: as in displacement addressing, but register r0 (zero) is used, e.g. `ldw r6, 100(r0)`

Nios II Programming Model Instruction Set

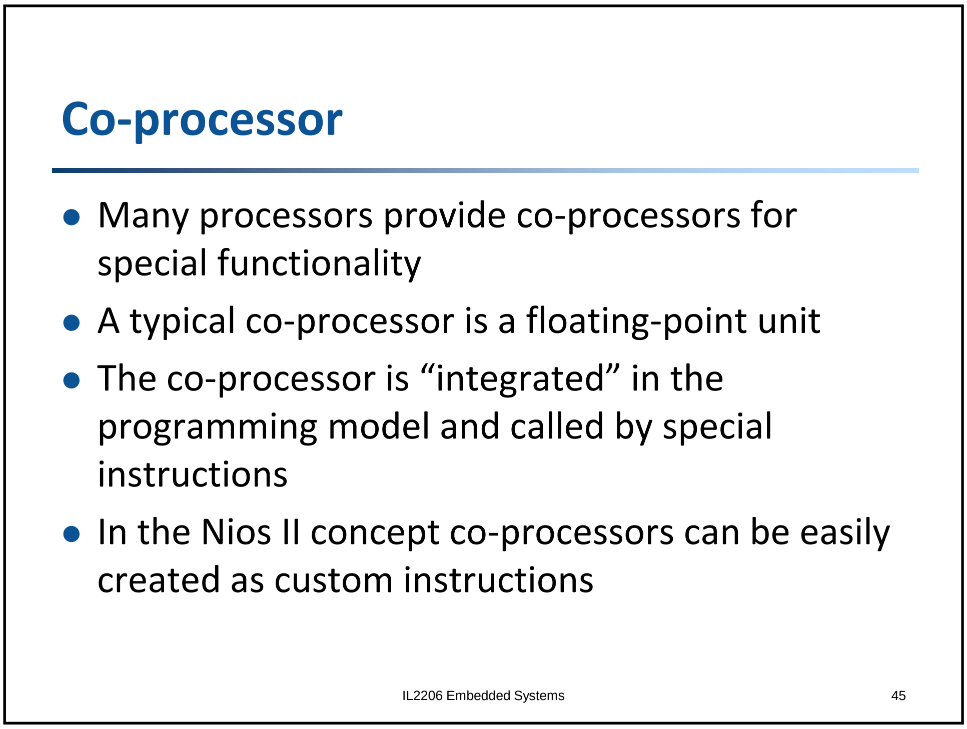
Data Transfer

- Nios II has a RISC (Reduced Instruction Set Computer) load-store architecture
- Load and Store instructions handle all data movement between registers, memory, and peripherals
- There are instructions for both cached (`ldw`, `stw`) and uncached access (`ldwio`, `stwio`)
- There are instructions that support half-word and byte transfers
- *Example*: `stw r5, (r9)` stores the value of word in `r5` in the memory location that is given by `r9`

Nios II Programming Model Instruction Set

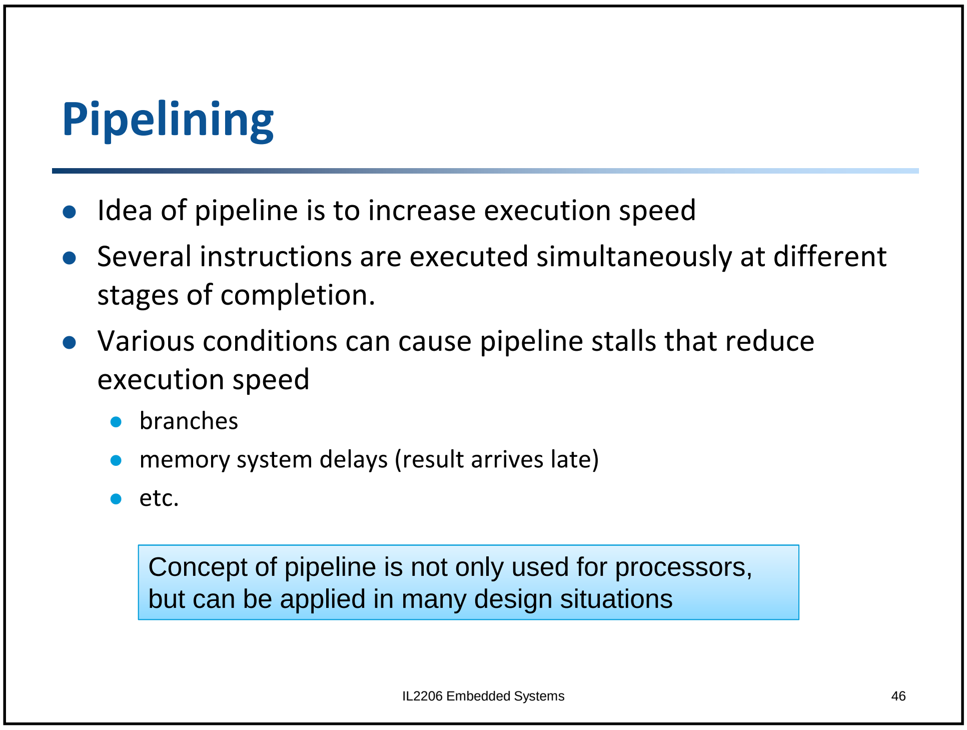
Custom Instructions

- Nios reserves instruction codes for custom instruction that designers create by defining the hardware for these functions
- Critical functions can be executed much faster



Co-processor

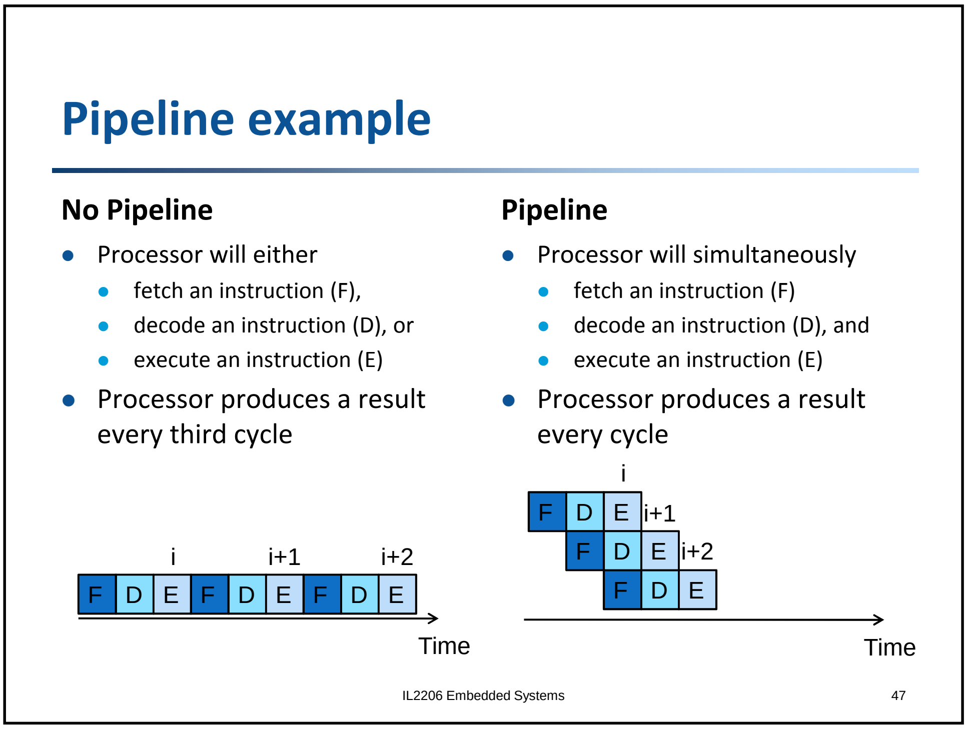
- Many processors provide co-processors for special functionality
- A typical co-processor is a floating-point unit
- The co-processor is “integrated” in the programming model and called by special instructions
- In the Nios II concept co-processors can be easily created as custom instructions



Pipelining

- Idea of pipeline is to increase execution speed
- Several instructions are executed simultaneously at different stages of completion.
- Various conditions can cause pipeline stalls that reduce execution speed
  - branches
  - memory system delays (result arrives late)
  - etc.

Concept of pipeline is not only used for processors, but can be applied in many design situations



Pipeline example

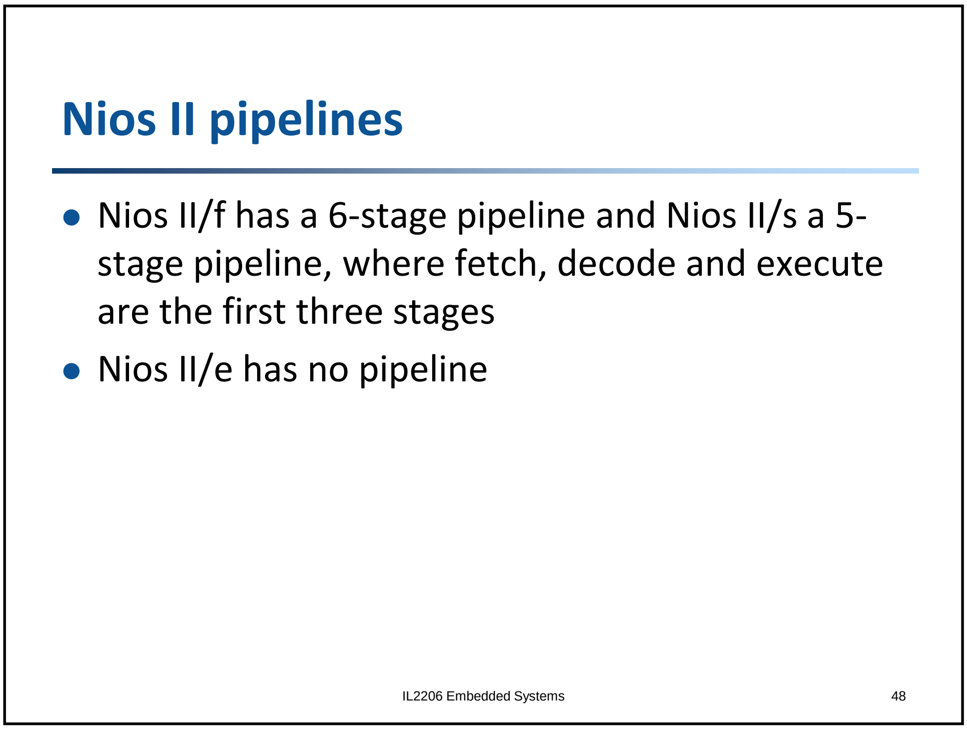
No Pipeline

- Processor will either
  - fetch an instruction (F),
  - decode an instruction (D), or
  - execute an instruction (E)
- Processor produces a result every third cycle

Pipeline

- Processor will simultaneously
  - fetch an instruction (F)
  - decode an instruction (D), and
  - execute an instruction (E)
- Processor produces a result every cycle

i
i+1
i+2
F D E F D E F D E
Time
i
F D E
i+1
F D E
i+2
F D E
Time



Nios II pipelines

- Nios II/f has a 6-stage pipeline and Nios II/s a 5-stage pipeline, where fetch, decode and execute are the first three stages
- Nios II/e has no pipeline

Latency and throughput

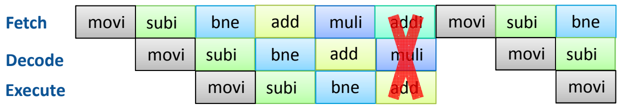
- Latency: time (or number of cycles) from entering the pipelining until it is completes
- Throughput: number of instructions executed per time period

Pipeline is used to increase **throughput**

Pipeline stalls

- If every step cannot be completed in the same amount of time, pipeline stalls
- Bubbles introduced by stall increase latency, reduce throughput

```
loop   movi r2, 5
       subi r2, r2, 1
       bne loop
       add r5, r3, r4
       muli r6, r5, 10
       addi r6, r6, 2
```



Fetch	movi	subi	bne	add	muli	~~addi~~	movi	subi	bne
Decode		movi	subi	bne	add	~~muli~~		movi	subi
Execute			movi	subi	bne	~~add~~			movi

Execution Time For Branch Is Not Constant!

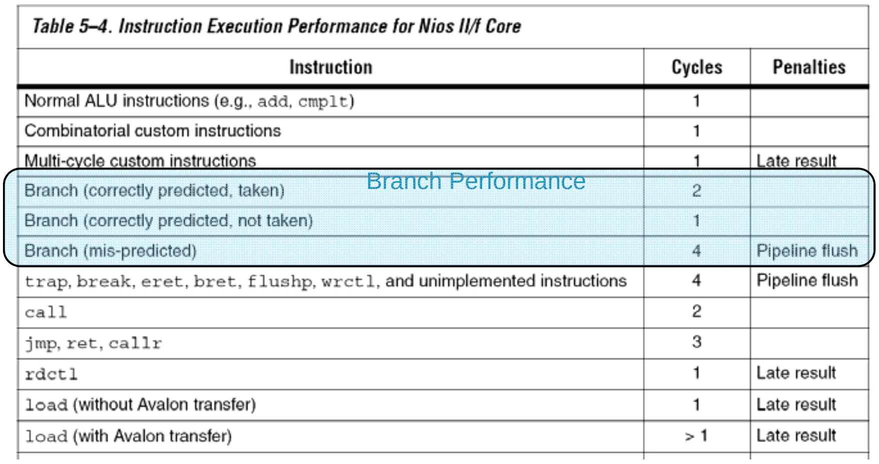


Table 5–4. Instruction Execution Performance for Nios II/f Core

Instruction	Cycles	Penalties
Normal ALU instructions (e.g., `add`, `cmplt`)	1	
Combinatorial custom instructions	1	
Multi-cycle custom instructions	1	Late result
Branch (correctly predicted, taken)	2	
Branch (correctly predicted, not taken)	1	
Branch (mis-predicted)	4	Pipeline flush
`trap`, `break`, `eret`, `bret`, `flushp`, `wrctl`, and unimplemented instructions	4	Pipeline flush
`call`	2	
`jmp`, `ret`, `callr`	3	
`rdctl`	1	Late result
`load` (without Avalon transfer)	1	Late result
`load` (with Avalon transfer)	> 1	Late result

Branch Performance

Multiplication/Division heavily depends on underlying hardware!

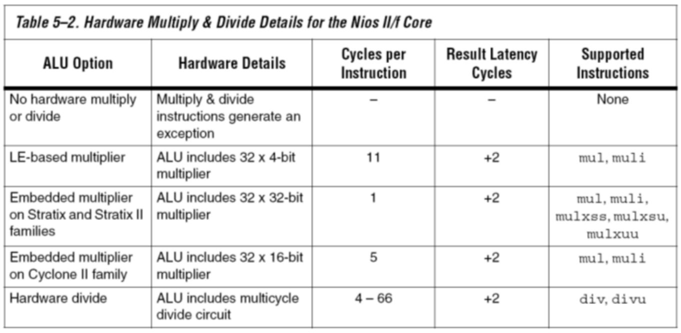
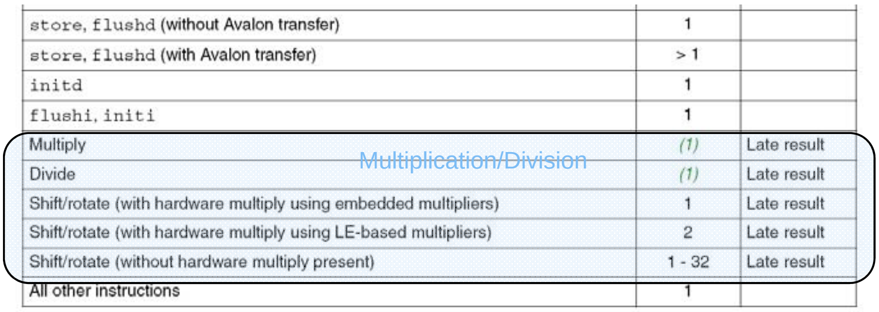


Table 5–2. Hardware Multiply & Divide Details for the Nios II/f Core

ALU Option	Hardware Details	Cycles per Instruction	Result Latency Cycles	Supported Instructions
No hardware multiply or divide	Multiply & divide instructions generate an exception	–	–	None
LE-based multiplier	ALU includes 32 x 4-bit multiplier	11	+2	`mul`, `muli`
Embedded multiplier on Stratix and Stratix II families	ALU includes 32 x 32-bit multiplier	1	+2	`mul`, `muli`, `mulxss`, `mulxsu`, `mulxuu`
Embedded multiplier on Cyclone II family	ALU includes 32 x 16-bit multiplier	5	+2	`mul`, `muli`
Hardware divide	ALU includes multicycle divide circuit	4 – 66	+2	`div`, `divu`

Execution Performance Nios II/f



Multiplication/Division

`store`, `flushd` (without Avalon transfer)	1	
`store`, `flushd` (with Avalon transfer)	> 1	
`initd`	1	
`flushi`, `initi`	1	
Multiply	*(1)*	Late result
Divide	*(1)*	Late result
Shift/rotate (with hardware multiply using embedded multipliers)	1	Late result
Shift/rotate (with hardware multiply using LE-based multipliers)	2	Late result
Shift/rotate (without hardware multiply present)	1 - 32	Late result
All other instructions	1	

Latency Cycles

- Latency Cycles can decrease performance, if data dependency exist
- Execution of the addi instruction is delayed by two additional cycles until the multiply operation completes

```
mul r1, r2, r3   #
addi r1, r1, 100 # (Depends on result of mul)
```

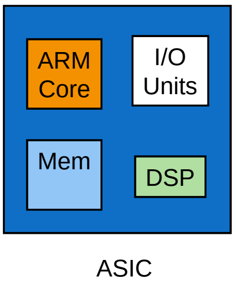
- In contrast, the code below does not stall the processor

```
mul r1, r2, r3
or r5, r5, r6    # No dependency on previous results
or r7, r7, r8    # No dependency on previous results
addi r1, r1, 100 # (Depends on result of mul)
```

Some words about the ARM Microprocessor Core

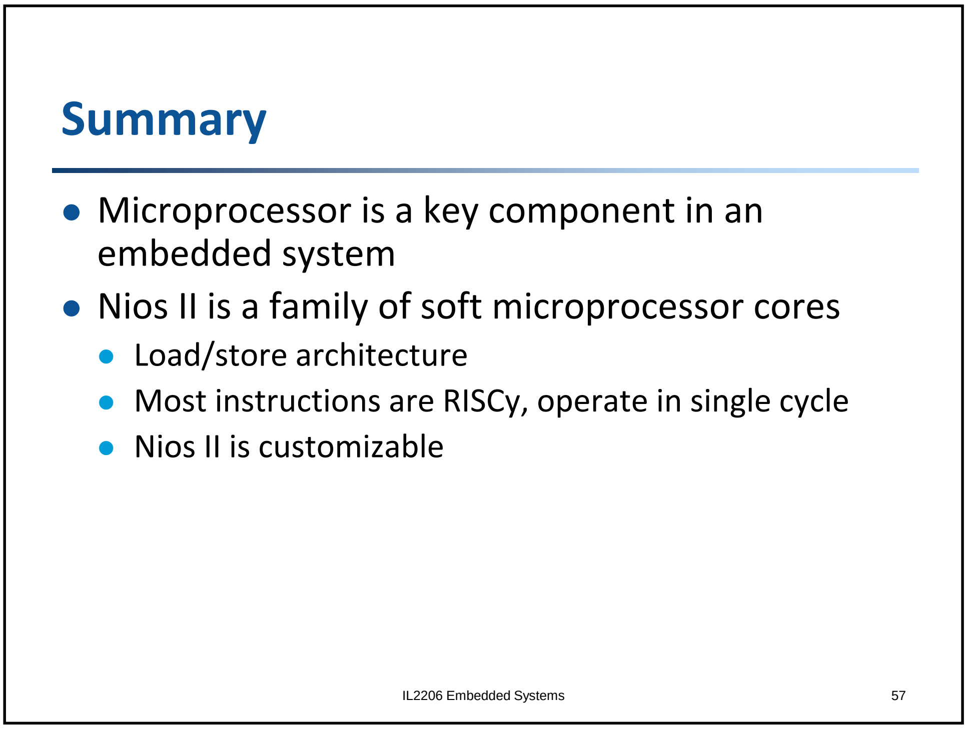
- ARM is a family of RISC architectures, which share the same design principles and a common instruction set
- ARM does not manufacture the CPU itself, but licenses it to other manufacturers to integrate them into their own system
- The ARM core is used in very many embedded products and can be seen as market leader

- The ARM core as part of a system-on-chip (in the same way as Nios II)



Further Reading

- Lee and Seshia, Introduction to Embedded Systems (Second Edition)
  - Chapter 8: Embedded Processors



Summary

- Microprocessor is a key component in an embedded system
- Nios II is a family of soft microprocessor cores
  - Load/store architecture
  - Most instructions are RISCy, operate in single cycle
  - Nios II is customizable