Embedded Systems

Microprocessors
Ver 2.0, 2021

Prof. William Fornaciari
Politecnico di Milano – DEIB
william.fornaciari@polimi.it

Outline

- Introduction
  - Microprocessor general purpose
    - CISC, RISC, Superscalar
    - CISC/RISC, EPIC/VLIW, CISC/VLIW
    - Comparison
  - Dedicated Microprocessors
    - Digital Signal Processors
    - Network Processors
    - Microcontrollers
  - Conclusions

Microprocessors

Introduction

Introduction

A Microprocessor is an electronic circuit that elaborates several instructions

- Part of the embedded applications implement in sw their algorithms
  - Flexibility, TTM, easy upgrade, sometime cost (dep on volume)

(During design process)

- Flexibility has two main flavors
  - **Maintainability and evolution of the application**
    - Sw development is less complex
    - Faster
    - Verification is less critical (even not trivial)
    - Sw designers are available in «volume»

Not only a blue sky

- Typically an equivalent hw solution has better performance than sw
  - Microprocessor architecture is optimized for flexibility but it is generic, not specialized
  - Performance is not only computing speed
    - Energy/power, Memory footprint, chip area + package, size, security, support by the suppliers, …

A typical 1st stemp in designing a system is similar to "reversing a design" that already exists and improving it

Example 1/2

- Sum of two 32-bit integers
  - Compare hw and sw realization, considering the availability of a 32 bit core for the sw execution. Same clock speed of Hw and Sw
  - Software implementation
    - Just one assembly instruction
  - Hardware implementation
    - Need to design and implement the entire 32 bit adder
  - It seems that the sw solution is the favorite one…but

Example 2/2

- New requirements: move from 32 to 33 bits, after an more accurate analysis of the required accuracy
  - Hw solution is almost the same in terms of architecture and performance
  - Sw solution require to use two o presumably three assembly instructions, with a performance penalty of around 100%

Introduction

- The «extreme» example, figures out the differences between a dedicated hw solution and a more flexible sw implementation
  - Designing a sw solution requires a deep knowledge of the architecture of the microprocessors available on the market
    - The characteristics of the problems (algorithm) should drive the designer, beside non functional constraints (performance, cost, development time, power, …)
  - Before to select a specific processer, the initial step is the choice of the «class» of processor to be used and in which «form» it has to be purchased

Classes of a processor

- Specialized vs generic
  - Specialized: *Application Specific Processor (ASP)*
    - Tailored for specific classes of applications like signal processing, image processing or the management of network data packets
  - Generic: *General Purpose Processor (GPP)*
    - 4-season processor, not optimized for any specific applications. Versatile, it can be used for different type of applications
  - Embedded Systems multi-processors
    - Typical solution is to use GPP for supervision and control of the activities of one or more ASPs

Typical for designing mobile applications

In which form is available?

- COTS vs IP
  - COTS (Component off the shelf) – just buy a chip off-the-shelf and mount in on a board (PCB) together with the interfaces to the rest of the systems Easiest way to create a system
  - IP (intellectual property) – the design (description) of the micro is purchased. Several abstraction levels full description using softwares "Verilog-like"
    - Soft-macro: the micro is described in HDL at RT-level
    - Hard-macro: description down to the level of layout Disclosed with lots of designs impossible to modify;they can be integrated with something else
  - Give the diffusion of PLD, the used of IP is becoming more and more popular. Lots of famous companies (ARM, Samsung, NXT...) adopt this solution
  - Typically the suppliers of CPLD and/or FPGA make available one or more cores (even for free)
    - Micro/Pico Blaze di Xilinx, Nios-II di Altera, Mico32 di Lattice, ARM7 di Arm Ltd. e Actel

Selection process

- Class
  - The main driver is the nature of the algorithm, the operations and data to be elaborated
- Form
  - The main driver is the target architecture of the systems
    - SoC, MPSoC, NoC, PCB, …
  - But performance and power, in addition to royalties, can make the difference

The choice can change during the development and availability

Selection process

- Performance
  - Main metric is the average number of instructions per clock cycle (IPC/CPI) Average can be misleading: out goal is to look to the peak
    - It is relative to the clock frequency, hence it need to be scaled to compare different architectures
  - MIPS- absolute measure of the throughput. It can be misleading when processors have different instruction sets (ISA)
    - MFLOPS is the same but for Floating Points operations
  - For more specific architectures, like DSP or NP, metrics can change
    - DSP, frequently it is used MMACS
    - NP, average number of processed packets in a time unit

Selection process

- Power is probably the most important driver for ES
  - Average power or peak power
    - Useful to roughly estimate the max power or the avg power of the overall systems

Average useful to have an idea regarding durability

Peak usefol to know the strenght

  - Frequently, especially at the beginning, it is used a joint measure between power and speed (related to performance)
    - For example mW/MHz or, at a higher level of abstraction, MIPS/mW
    - These information are very useful but must be properly used during the design process

Memory

Frequently underestimate, but it's a crucial topic. In Embedded Systems, we generally evaluate memory in terms of Kb. We use single chip apps, with a specific size

- Memory requirements (bandwidth and size) of the application can be a hard constraint
- In some case it could be impossible to use an external memory, only that internal to the micro is available
- Maybe that some bandwidth can be achieved only by integrated memories
- For complex systems, the crucial point could be the addressing space
  - Simple architectures exploits 16-bit addressing (64KB), 20 bits (1MB) or 24 bits (16MB), while only those more complex can arrive to 32 bits (4GB)

Peripherals

Something "hidden". Generally focus is on executing a code, but in Embedded Systems we have to catch signals from the external environment. That's why peripherals are needed

- An embedded systems typ elaborates external signals and control physical apparatus
- It is possible to select and architecture oriented toward the computing, and to design the rest of the system, or to look for a single chip solution integrating computing, interfacing and some peripherals
  - Microprocessor vs. Microcontrollers
    - The choice of the micro require a PCB or a SoC integrating all the peripherals under the responsibility of the designer
    - Using a microcontroller, the designer get rid of the integration problem, paid in terms of reduction in the number of alternatives

Available Software is crucial

- Many embedded applications have a limited legacy code, most of them exploit standard functions and libraries
  - E.g. numerical functions (FFT, Wavelet, …), Image processing (convolutional filters, motion detection, DCT, …)
- The availability of libraries simplifies both design and validation of the applications, making feasible in some cases the development of sw solutions otherwise impractical
- The availability of an operating systems and SDK for the specific processor is important, too

Software Development

- It is the cornerstone of the sw development, many differences exist among the various SDKs
  - Important aspects to be evaluated are the software compilation flow, the quality of the code generated and the flexibility and knobs offered to the designer
  - Concerning the analysis tools, the richness, their accuracy and reliability are important. Debuggers should be carefully analyzed
  - Documentation and reference design
  - Community end ecosystem

Something more tangible

- Packaging
  - If the micro is a COTS, different packages are available
  - Size, pinout, material (e.g. plastic, metal), certification (consumer, industrial, aerospace, …)
- Certifications
  - There exists several certification
    - Consumer, industrial, aerospace, automotive, military, …
      - Some are specifically tailored for narrow fields
        - » *Automotive: MISRA rules (http://www.misra.org.uk/)*
  - The availability of a component with a proper certification could be a must constraint for its adoption

Microprocessors

General Purpose Processors

GPP

GPP

- GPP - generic architecture valid for a range of application fields Slow interface with the environment (use of sensors)
  - Common in PC/PDA
    - For ES low end when energy and performance are not crucial and the nature of the application is quite heterogeneous
    - Control and management of a slow interfacing with sensors, interactivity via GUI
- GPPs can have different architectures
  - CISC vs RISC
    - CISC: huge amount of complex instructions
    - RISC: few simple instructions prepared for Embedded Systems

GPP

CISC

GPP

- CISC
  - The first GPP were designed by following the approach of a *Complex Instruction Set Computer*
  - Limited amount of memory at that time
  - Need to obtain compact code, with few lines
    - ISA and architectures designed to carry out three operations (*load*, *execute* e *store)* in a single instruction
    - The addressing mode has been enriched
    - Development of the compiler simplified

GPP

- Architetture CISC
  - Addressing modes

Modalità	Esempio	Significato
Assoluto (istruzioni)	`jmp a`	PC ← a
Relativo (istruzioni)	`jmp (a)`	PC ← PC + a
Indiretto (istruzioni)	`jmp Rs`	PC ← PC + Rs
Registro	`add Rd,R1,R2`	Rd ← R1 + R2
Assoluto	`load Rd,a`	Rd ← M(a)
Indiretto	`load Rd,[Rs]`	Rd ← M(Rs)
Base, offset	`load Rd,a[Rs]`	Rd ← M(Rs + a)
Indice	`load Rd,[Rs,Ri]`	Rd ← M(Rs + Ri)
Indice, offset	`load Rd,a[Rs,Ri]`	Rd ← M(a + Rs + Ri)
Indice, offset, scala	`load Rd,a[Rs,Ri],s`	Rd ← M(s × (a + Rs + Ri))
Autoincremento	`load Rd,+[Rs]`	Rd ← M(Rs), Rs ← Rs + 1
Autodecremento	`load Rd,-[Rs]`	Rd ← M(Rs), Rs ← Rs − 1
Push	`push Rs`	M(SP) ← Rs, SP ← SP + 1
Push	`pop Rd`	SP ← SP − 1, Rd ← M(SP)

GPP

- CISC
  - Ideally, each Arith/Logic operation should be capable to access (read) data and write the result by exploiting any of the available addressing modes
    - Fully *orthogonal* architecture and Instruction Set
  - In the reality, the number of operation-addressing mode combinations has been limited
    - Usually one of the operand refers to the memory, while the other is a register with the role of both source and destination
    - Despite this simplification, CISC instructions are coded in variable length formats
      - More complex fetch and decode units and need of more speed when accessing the memory

GPP

- CISC Architecture
  - The ALU, is implementing a variety of the basic operations
    - Add/Sub, mul, div, logic operations, shift, comparison
    - In some architectures the instructions can be vectorial, working on more registers at the same time
      - *Single Instruction Multiple Data*
  - So many addressing mode and A/L operations, require a complex *data-path* that is hard to make fast

GPP

- CISC Architectures
  - Some flavors of the ADD operation

Istruzione	Significato
`add`	Istruzione di base
`adc`	Somma con riporto in ingresso
`addi`	Somma con una costante
`addi`	Somma con una costante a 32 bit
`addq`	Somma con una costante a 3 bit
`aaa`	Somma di operandi unpacked BCD
`daa`	Somma di operandi packed BCD
`xadd`	Somma e scambia gli operandi
`addps`	Somma di operandi a 128 bit
`addss`	Somma dei 32 LSB di operandi a 128 bit
`paddb`	Somma i singoli byte
`padds`	Somma i singoli byte con saturazione
`adawi`	Somma operandi allineati alla parola
`addb3`	Somma tre operandi a 8 bit
`addw3`	Somma tre operandi a 32 bit
`addl3`	Somma tre operandi a 64 bit

GPP - CISC

- To increase the *throughput*, the CISC processors implement complex pipeline (e.g. up to 23 stages) that are hard to manage

- Another strategy, still very costly in terms of hw resources, is based on altering the order of execution of the assembly instruction without modifying the semantics of the program
  - For a compiler, when considering so complex pipelines, it is hard to control the status of the execution of the instruction
    - The instructions could arrive to the decoding in an order that is not optimal w.r.t. the status of the pipelines at that time
  - Modern CISCs have dedicated hw units to dynamically reschedule the instructions
    - Out-of-order execution
  - CISC have remarkable performance but, over the time, energy efficiency and cost moved in favor of other architectures

GPP

RISC

GPP

- RISC - *Restricted Instruction Set Computer*, when the architecture has few simple instructions
  - End of '80 the price of DRAMs fall down
    - Size of the program is no longer a wall
      - » CISC operations can be decomposed into RISC instructions to simplify the architecture
  - In the same period the integration scale raised up, dramatically
    - Possibility to realize on a single chip complex architectures
    - Pipeline and quantity of GP registers
    - To exploit pipelining, the execution time of the instructions must be uniform
    - This is possible when instructions are simple

GPP

- RISC
  - Main advantage is simple instructions set and simple architecture
  - Instruction decoding is simplified
    - Frequently RISC instruction have fixed lengths
    - Easy balancing the stages of the pipeline and to increase clock
  - Execution units (ALU and BPU) take advantage from simple inst
    - RISC operates only on registers hence more speed and duration is predictable
      - RISC processor tends to have many registers, so the presence of memory spill is reduced significantly

GPP

- RISC
  - I/O, namely load/store from memory/registers and viceversa
    - Each instruction requires to pre-load of the date and, eventually, to write the result in memory
      - **load/store architecture**
  - Data transfer operation are crucial for performance
    - Fortunately, the instructions have a limited number and simple of addressing modes
    - Aggressive use of memory (cache) hierarchy to reduce the access time to data

GPP

- RISC
  - Addressing modes

Modalità	Esempio	Significato
Relativo (istruzioni)	`jmp (a)`	`PC ← PC + a`
Indiretto (istruzioni)	`jmp Rs`	`PC ← PC + Rs`
Registro	`add Rd,R1,R2`	`Rd ← R1 + R2`
Base, offset	`load Rd,a[Rs]`	`Rd ← M(Rs + a)`
Autoincremento	`load Rd,+[Rs]`	`Rd ← M(Rs), Rs ← Rs + 1`
Autodecremento	`load Rd,-[Rs]`	`Rd ← M(Rs), Rs ← Rs − 1`
Multiplo	`load {R1,R2,...},[Rs]`	`R1 ← M(Rs), R2 ← M(Rs+1), ...`

GPP - RISC

- Software development flow
  - Considering the same high level source code, the number of RISC instructions will be higher that CISC
  - RISC architectures has no hw units to solve pipeline conflicts
  - But good predictability of execution allow the compiler to generate efficient code
- Power consumption
  - RISC are simple and inherently low power w.r.t. CISC
    - 20-100 mW/MHz or around mW/MIPS

GPP

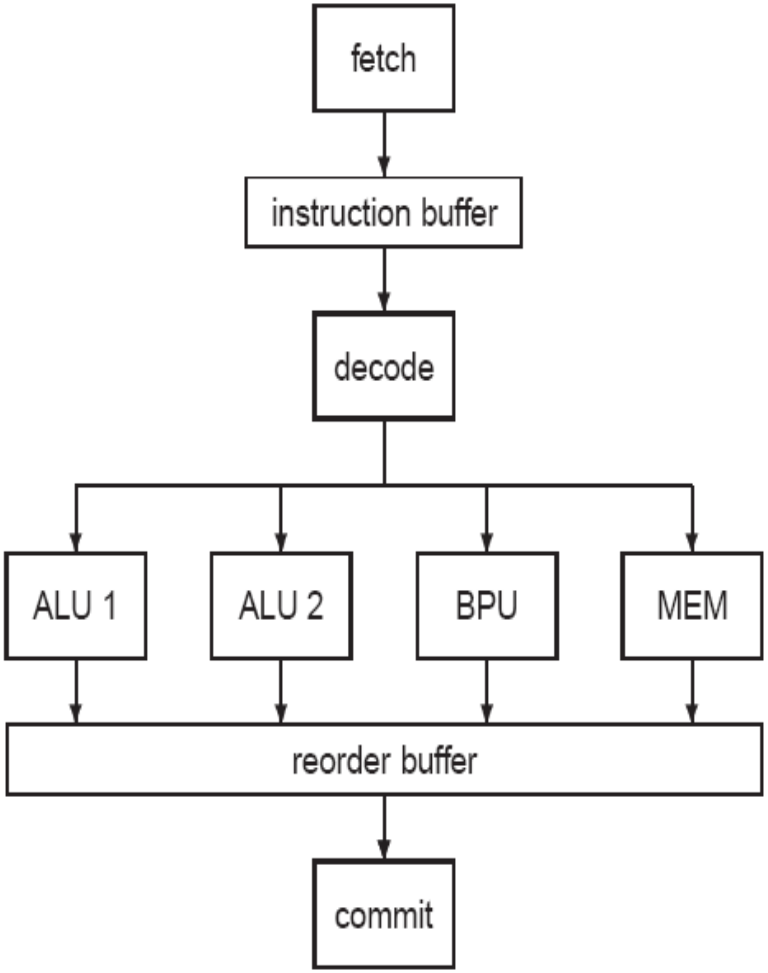
Superscalarity

GPP

- Superscalar architectures
  - There exists a lot of parallelism at the level of assembly instruction
    - Example: y=a+b and z=c*d
      - There is no dependency, hence any execution order is fine or they can execute in parallel
    - This consideration spurred the appearing of the superscalar architectures
  - It is superscalar an architecture having more than one executing units, hence more ALUs and or BPU

GPP

Typical super scalar architetcure



GPP – super scalar

- The parallelism offered by the execute stage allows to deliver more instruction per clock cycle
  - The complexity of the control logic increases as the impact on clock speed and power consumption
  - Moreover, it is the microprocessor to define dynamically the scheduling of the instructions
    - Pros
      - Compiler has not to work on organizing the code for a specific hardware structure
    - Cons
      - A relevant part of the processor is deserved to implement complex scheduling mechanisms and to manage the consistency

GPP

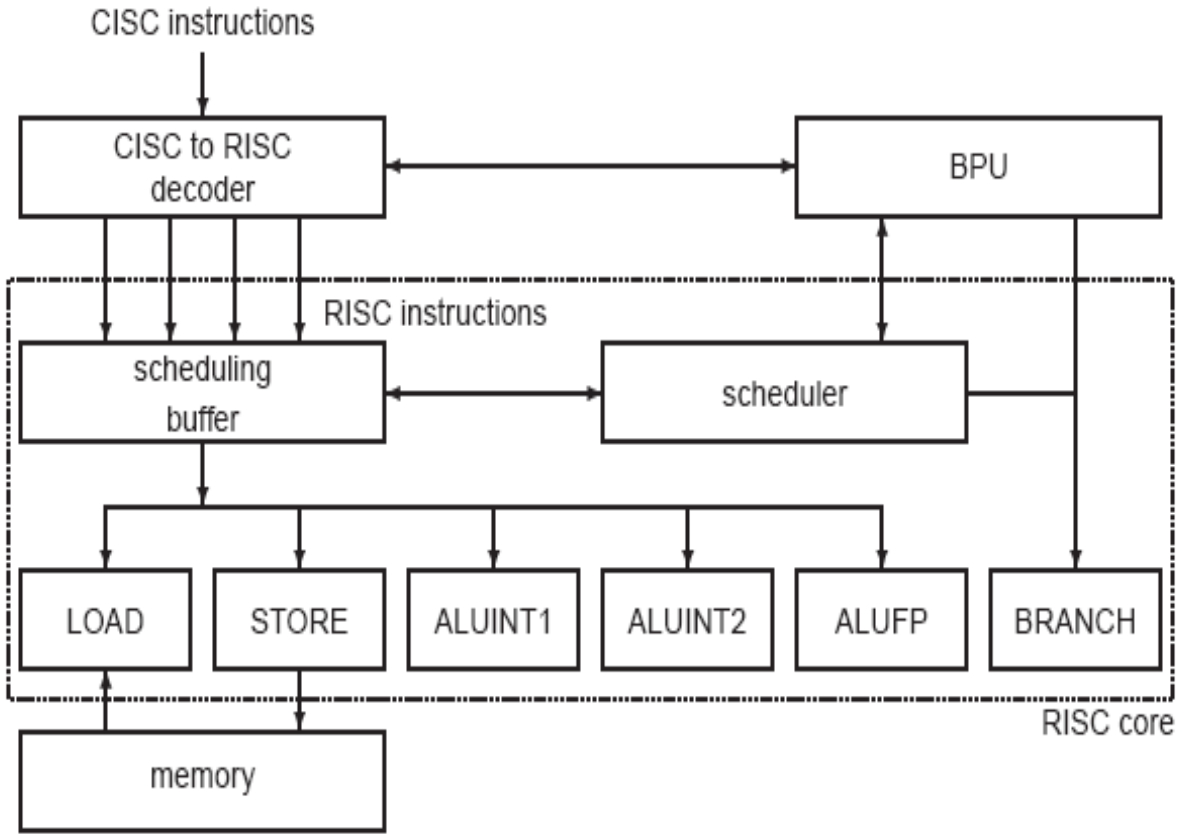
CISC/RISC

GPP

- CISC/RISC
  - Approach combining *out-of-order execution* and *superscalarity* with the goal to remain compatible with code developed in the past
    - Instruction set x86 is the standard de-facto for CISC
  - Each CISC instruction is decomposed in a set of simple instructions easy to be executed by a simple and efficient core
    - There is an hardware cost, but this solution joins the benefit of backward compatibility with the high-performance achievable by a RISC core

GPP

- CISC/RISC
  - Structure of a CISC based on a RISC core



GPP

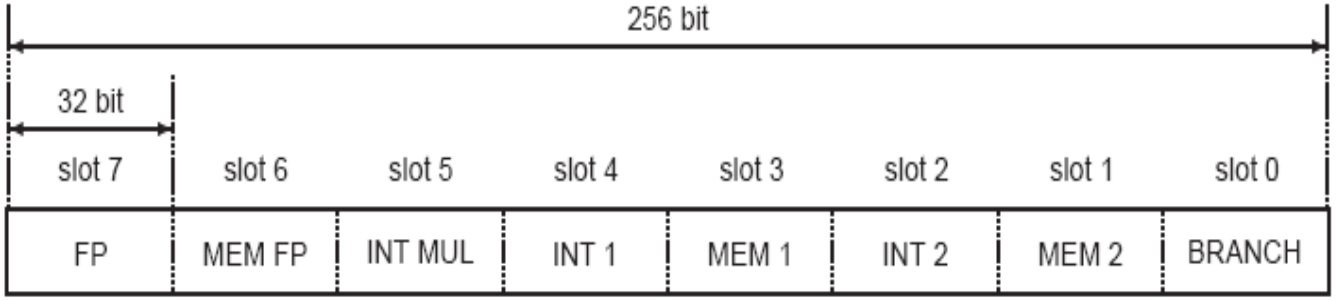
EPIC/VLIW

GPP

- EPIC/VLIW
  - To overcome the limits related to the complexity of the hw scheduling of the instructions, Intel developed in the past a new class of processors named EPIC (or VLIW). Itanium has been the first commercialized
  - These architectures support explicit parallelism
    - Possibility to execute more instructions at the some time under the control of the program
      - A VLIW instruction is the packing of more elementary instructions, typically RISC, in a single wide word, having a fixed structure

GPP

- EPIC/VLIW
  - Structure of a VLIW instruction

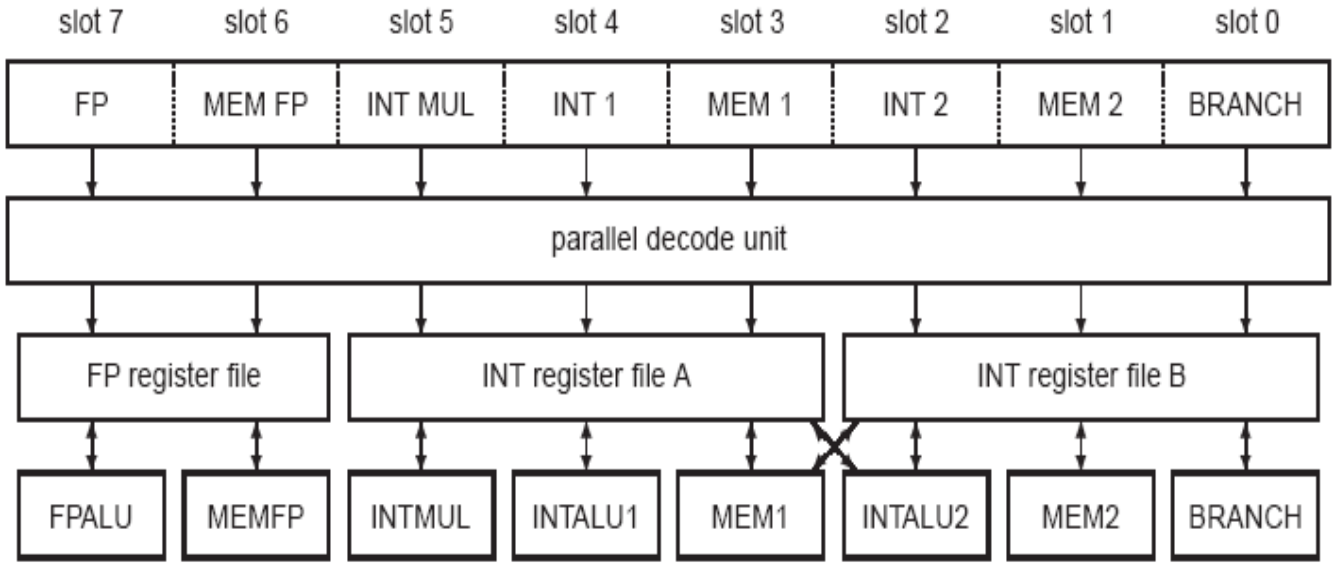


GPP

- EPIC/VLIW
  - The VLIW word, once fetched from the memory, is decoded in parallel
    - At the end of the decoding, the single RISC instructions corresponding to the various slots, are dispatched to the different execution units
  - This fixed structures moves to the compiler the task to perform a complex scheduling
    - The algorithms to be optimized should be well structure to fit well a specific architecture

GPP

- EPIC/VLIW
  - To execute in parallel more instructions, the register files should be equipped with multiple read ports and should be connected flexibly to the different units



GPP

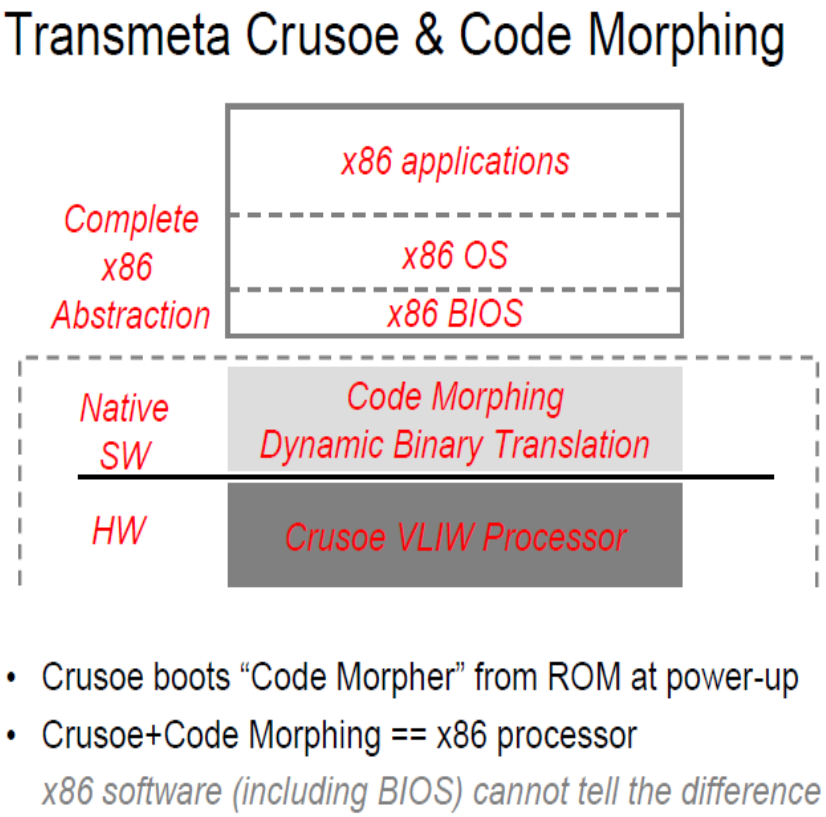
- EPIC/VLIW
  - EPIC/VLIW has the goal to move the complexity of the scheduling and code optimization towards the compiler
    - The architectures are simple, even if of large size given the number of execution units
  - Power consumption is in between simple and superscalar RISC
    - Similar solutions, allowing to achieve peak performance, has been adopted also in non general purpose architecture like the *digital signal processors (DSPs)*

GPP

CISC/VLIW

GPP

- CISC/VLIW
  - Similar to CISC/RISC, performance of VLIW with the goal of backward compatibility
  - CISC instructions are decomposed in terms of basic ones execute by a VLIW (simple and efficient) core
    - Example (one one?)
      - Transmeta Crusoe
      - Today available only as IP



GPP
Comparison

	RISC	CISC	Superscalari	EPIC/VLIW
Instruction set	semplice	complesso	complesso	semplice
Modi d'indirizzamento	pochi	molti	molti	pochi
Dimensione istruzioni	fissa	variabile	variabile	fissa, grande
Register file	singolo[1]	singolo[1]	singolo[1]	multiplo
Numero registri	alto	basso	medio	molto alto
Scheduling istruzioni	statico	dinamico	dinamico	statico
Gestione conflitti	statico[2]	dinamico	dinamico	statico
Complessità compilatore	media	alta	bassa	molto alta
Parallelismo	assente	assente	implicito	esplicito
Complessità hardware	bassa	alta	molto alta	alta
Complessità pipeline	bassa	alta	molto alta	media[3]
Consumo di potenza	molto basso	alto	molto alto	alto
IPC tipico	≈ 1	< 1	> 1	$\gg 1$

[1] In alcuni casi disgiunto tra registri interi e registri floating-point.

[2] I conflitti vengono riconosciuti e risolti unicamente mediante stalli.

[3] Struttura semplice, ma di grandi dimensioni per via delle molte unità di elaborazione.

Microprocessors

Application Specific

ASP

- ES need to solve problem with high degree of specialization and with a limited number of functions
  - GPP are frequently not the best choice
- Specialized architectures have been designed
  - Differences in terms of instruction sets, architecture, size, power consumption, computing power, …
    - Digital Signal Processor, Network Processors, Microcontrollers

ASP

Digital Signal Processor

ASP

- Digital Signal Processor
  - DSP are the popular one dedicated processors
    - They can vary a lot among the producers, but they have in common the design tailored to perform numerical computing
  - To understand their structure, let's review the main operations used in the algorithm for numeric processing

Operazione	Forma chiusa	Forma iterativa
Somma vettoriale	$\mathbf{Z} = \mathbf{X} + \mathbf{Y}$	$z_n = x_n + y_n \quad \forall n$
Riscalamento	$\mathbf{Z} = k\mathbf{X}$	$z_n = k \cdot x_n \quad \forall n$
Somma	$z = \lvert\mathbf{X}\rvert_1$	$z = \sum_n x_n$
Somma quadratica	$z = \lvert\mathbf{X}\rvert_2$	$z = \sum_n x_n^2$
Prodotto scalare	$z = \mathbf{X} \times \mathbf{Y}$	$z = \sum_n x_n \cdot y_n$
Convoluzione	$\mathbf{Z} = \mathbf{X} * \mathbf{Y}$	$z_n = \sum_k x_k \cdot y_{N-k} \quad \forall n$
Trasformata di Fourier	$\mathbf{Z} = \mathcal{F}(\mathbf{X})$	$z_n = \sum_k x_k e^{\frac{-2\pi i}{N} nk} \quad \forall n$

ASP

- Digital Signal Processor
  - Widely used operation

$$z_{t+1} = z_t + x * y$$

    - It is so important for DSP to worth a specific name
      - ***MAC : Multiply Accumulate***

  - A valuable DSP architecture should optimized as much as possible the sum and mul operation, possibly by exploiting a single 3-operand instruction

ASP

- Digital Signal Processor
  - Given the nature of the operations, a relevant part of the algorithms to be executed, contains loops
  - The possibility to optimize loops in a DSP is possible, since
    - The body is typically small (~10 asm instructions)
    - The control loop variable is not altered within the loop body
    - The update of the control loop variable is simple
    - The pattern to access the vector is generally regular and simple
  - Specific Hw to optimize loops
    - Circular buffer to store the loop body
    - Dedicated register where to store the control loop variable
    - Add/Sub to perform inc/dec operation without using ALU

ASP

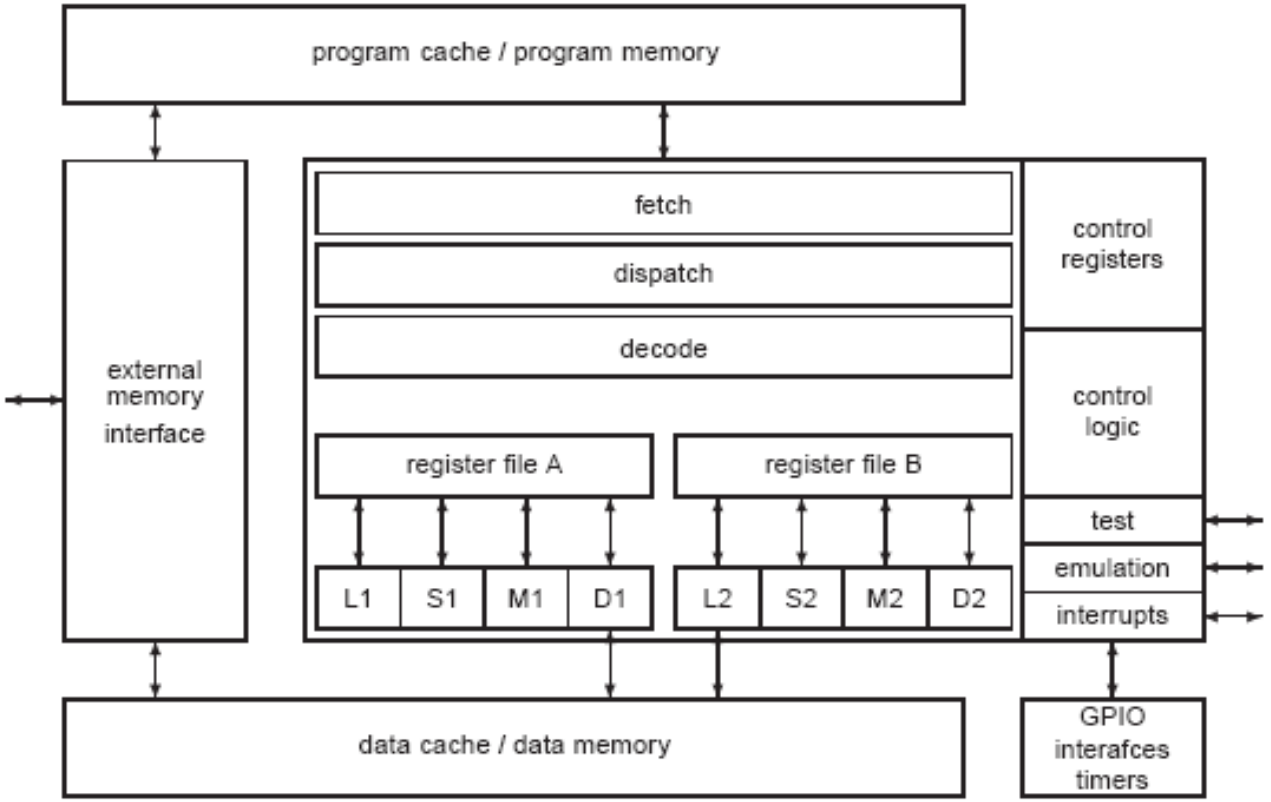
- Digital Signal Processor
  - Access to the data in memory could be a bottleneck
    - High bandwidth interfaces are designed to this purpose
  - Memory hierarchy for DSP
    - High speed bus with word width up to 256 bits
    - Multi-level cache hierarchy
      - Classic
        - » Stanford: unified for data and instructions
      - Specialised
        - » Harvard: cache L0 separate for data and instructions
        - » SHARC: Three L0 caches for data, instructions and constants

ASP

- Digital Signal Processor
  - In addition to the flexibility and computing power offered by the special instructions and by the specific ALUs, modern DSPs are frequently organized by following a VLIW approach
  - In fact, it is suitable for numerical applications, where the control is limited and there exists a high intrinsic parallelisms of operation

ASP

- Digital Signal Processor
  - Example: DSP VLIW by TEXAS



ASP

Network Processors

ASP

- Network Processor
  - Dedicated architectures tailored to process network packets
    - Complex system-on-chip with high parallelism, for network applications (e.g. routers)
  - Functionalities and protocols are evolving, the design of a NP is not trivial
  - Classes of the typical operations of a NP
    - Buffering of input/output packets
    - Remove/add of header at the level of data-link
    - Search for a specific field in an IP header
    - Search for an address in a lookup table
    - Computation of CRC codes
    - Send or delete of packets

ASP

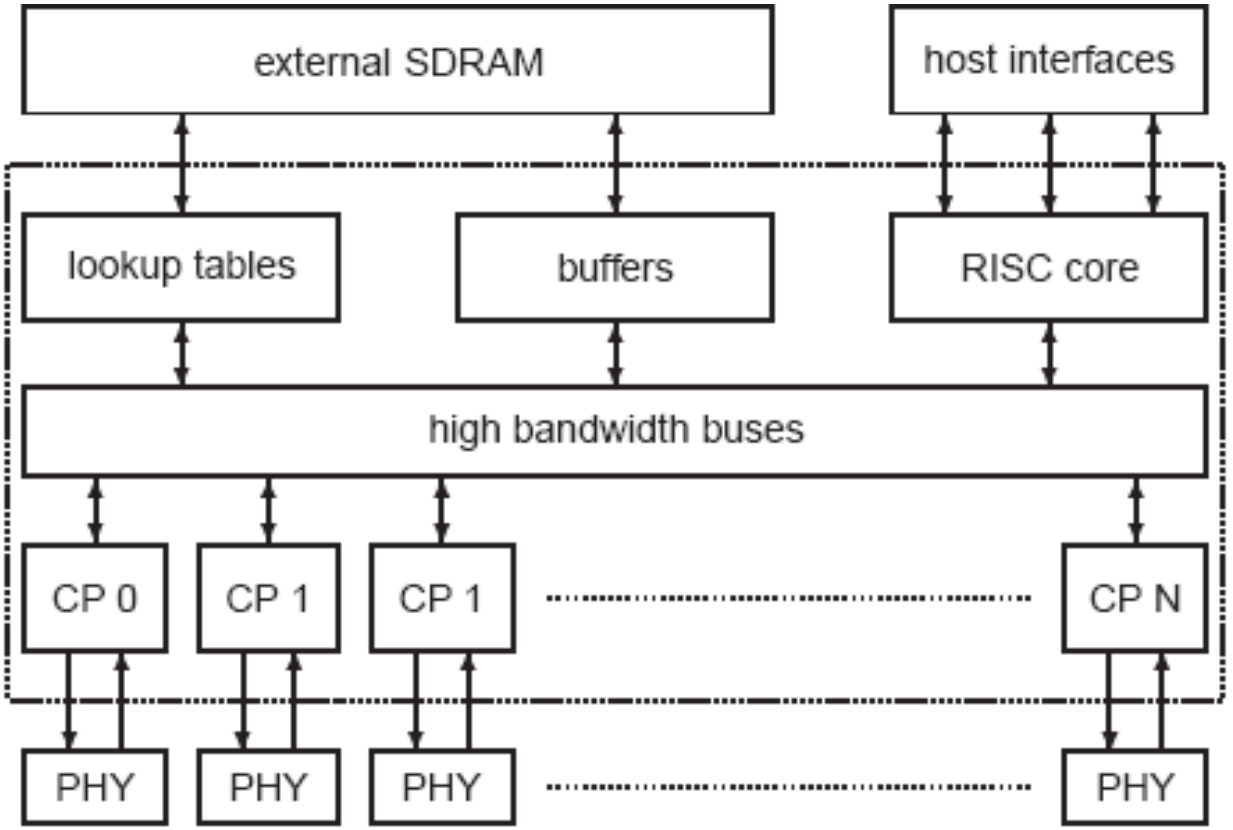
- Network Processor
  - To increase performance, dedicated hw is used
    - Fast I/O interfaces, queue management, cripto-cores, multi RISC cores
  - Frequently packet processing application requires the management of several independent channels who can benefit from parallel processing
  - Each channels is managed by a dedicated hw units frequently named as PP (*Packet Processor*) or CP (*Channel Processor*)

ASP

- Network Processor
  - When the separate flows and not fully disjoint
    - Use of RISC cores, to supervise and for high-level management
  - The typical structure of an application of *packet processing* is a set of simple routines, each working on a huge amount of packets
    - Small programs working on a enormous amount of data

ASP

- Network Processor
  - Typical architecture of a NP



ASP

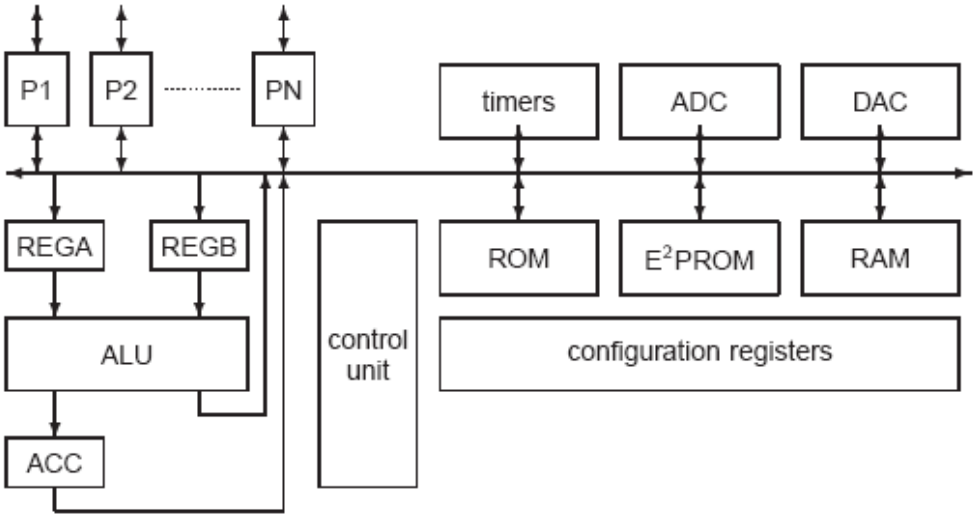
- Network Processor
  - Which type of programming?
    - Core RISC
      - Development of programs starting from C source code like that of a RISC
    - Single PP
      - Sometime is still carried out in assembly, bunch of simple routines
        - » Programming CP is not well structured and only some time we can use a functional approach based on pattern recognition

ASP

Microcontrollers

ASP

- Microcontrollers
  - (MCU – *Micro Controller Unit*) is a class of microprocessor integrating peripherals and interfaces on a single-chip
  - Their SoC nature is fine to deal with not top computing power, but with application having constraints on the use of hw resources and development time



ASP

- Microcontrollers
  - Frequently there is no interface to external memory
    - Programs for MCU are typically small
    - Memory interface increases the pinout
  - Small size in frequently a strict requirement
    - To provide a memory interface limiting the pinout, some architectures use a multiplexing of the I/O ports
      - The control logic must be properly configured to exploit this operating mode

ASP

- Microcontrollers
  - The architectures offer several peripheral and interfaces
    - I2C, SPI, CAN, JTAG, *PWM*, *UART/USART*, *watchdog*, *timer*, analog I/O and comparators, etc…
  - Programming in typically in C, only sometime in assembly
  - SDKs for microcontrollers ranges from simple C compiler up to integrated framework for performance analysis and configuration management