UNIVERSITÀ
DEGLI STUDI DI TRIESTE

dipartimento
di ingegneria
e architettura

Control unit

A.Carini – Microcontrollers

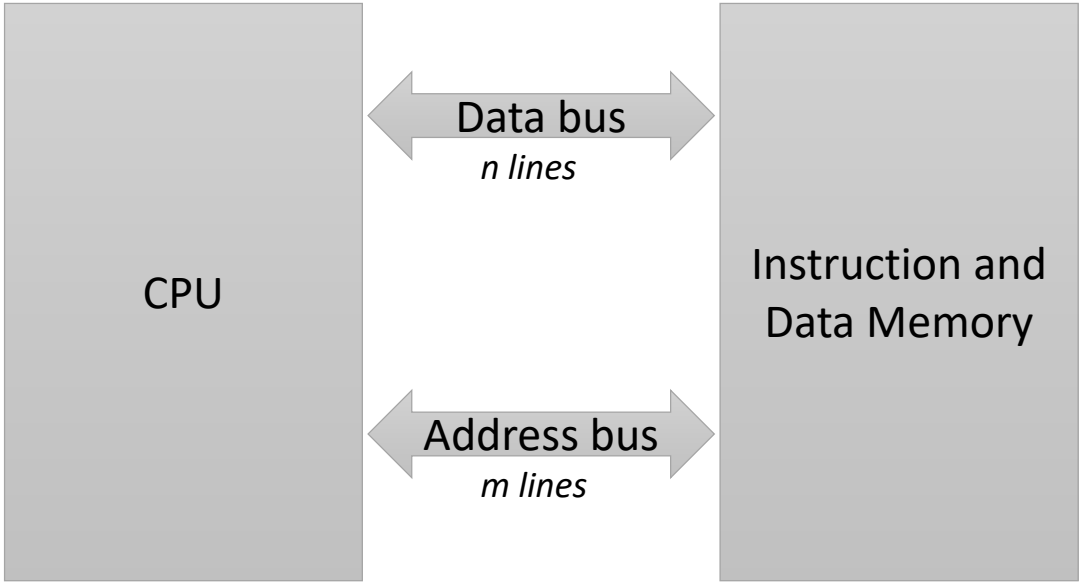
Control unit

- The control unit is composed by all those circuits that ensure the right operation of the CPU.
- First of all, these circuits allow to *fetch, decode, execute* the instructions of a program.
- Moreover, they allow to *manage* the *external peripherals*, like the memory, the I/O data channels, permitting the processor to interact with the external world when required.
- The design of a CPU control unit is strictly related to its internal hardware organization, and to its architecture, meaning the available set of instructions and the data addressing modes to be provided to the programmer.

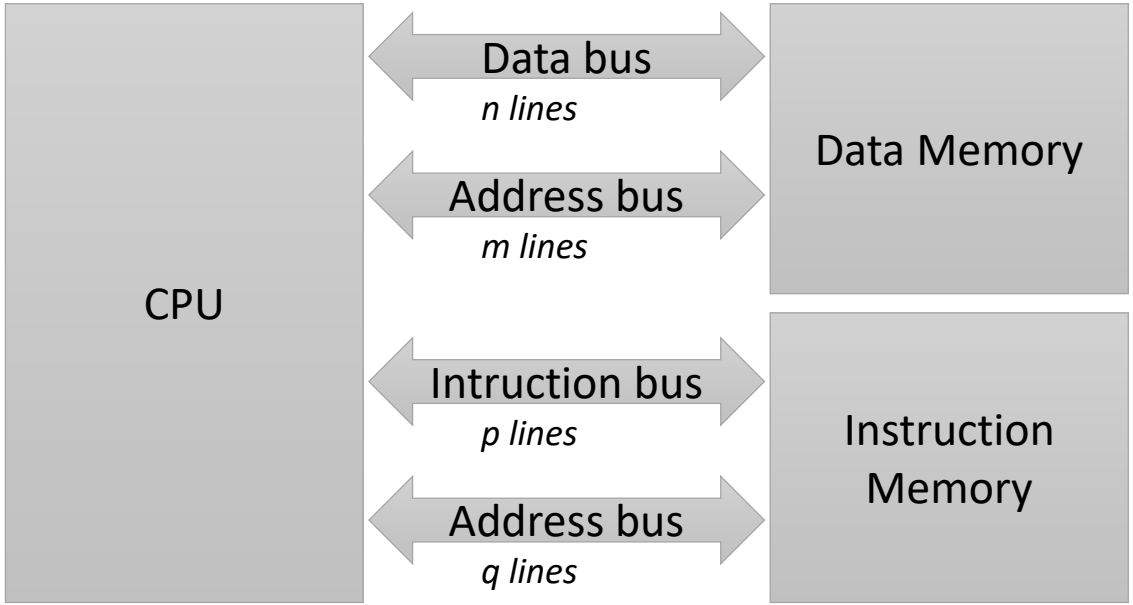
Microprocessor organization

- Two main strategies:
- *Von Neumann* organization
  - simple and cheap, used in general purpose processors (GPPs) and low-cost microcontrollers (µCs) (but also in ARM 7).
  - Data and instructions share the same physical memory. Thus, only a bus.
- *Harvard* organization
  - More complex and costly, used in high performance processors and DPSs.
  - Data and instructions are stored in separate physical memories. There are two or more bus systems.
  - Introduces a first degree of parallelism: can fetch an instruction and its operands in the same clock cycle, reading two different memories.
  - *Modified Harvard* introduces more parallelism: some functional units (e.g., ALU) are replicated in order to allow parallel execution of instructions.

Von Neumann architecture



Harward architecture



Bus system

- Inside a processor, buses allow data transfer between different functional units.
- They are composed by a certain number of lines, *bits*, that can assume two logic levels.
- Buses are driven by suitable logic gates, characterized by a *three-state* output. Thus, the unit can be disconnected from the bus at the end of data transmission.
- Different types of buses are present in µCs and DSPs. As in GPPs, separate buses transfer data/instructions and addresses. There are buses for interrupts and internal buses for specific functional units. Their number bits can be different.

Example

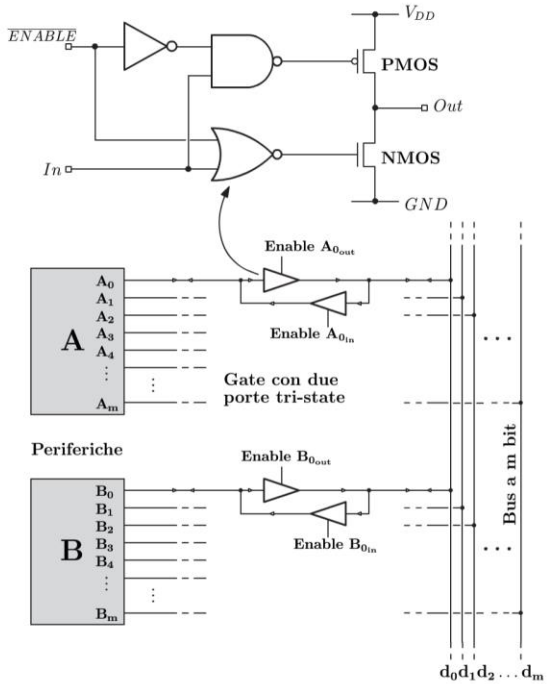


Figura 5.3: Connessione bidirezionale di due periferiche in una CPU attraverso un bus a m bit. Si noti l'uso di gate di connessione basati su porte tri-state, di cui si fornisce uno schema di principio.

Example

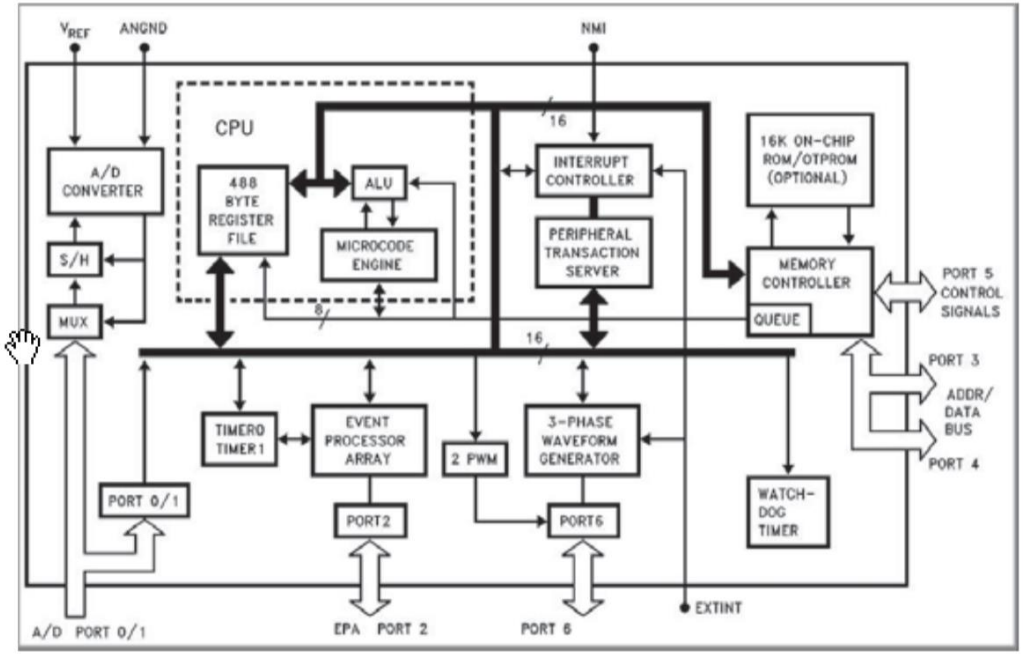


Figura 5.4: Organizzazione interna di un microcontrollore Intel 80C196. Il bus è unico, secondo il paradigma di von Neumann.

Example

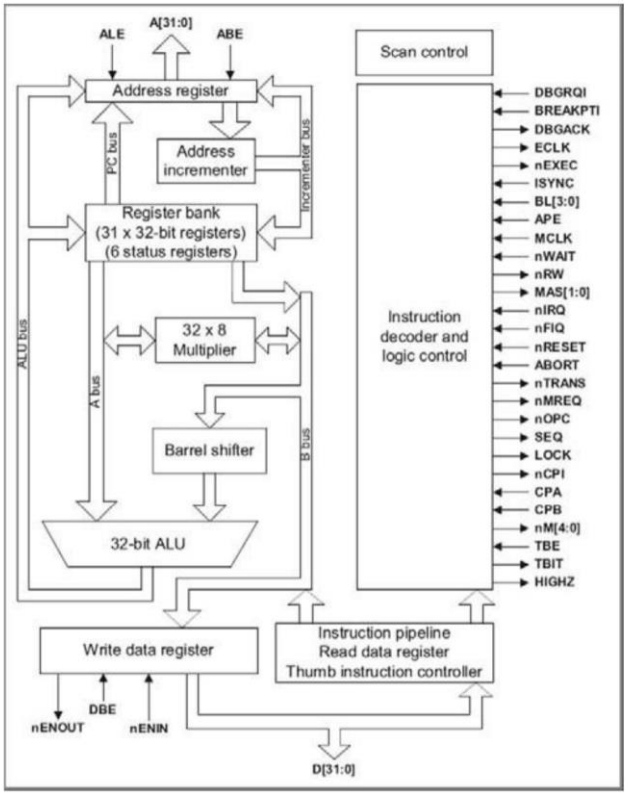


Figura 5.5: Organizzazione interna di un processore ARM7©. Il bus è unico, secondo il paradigma di von Neumann.

Example

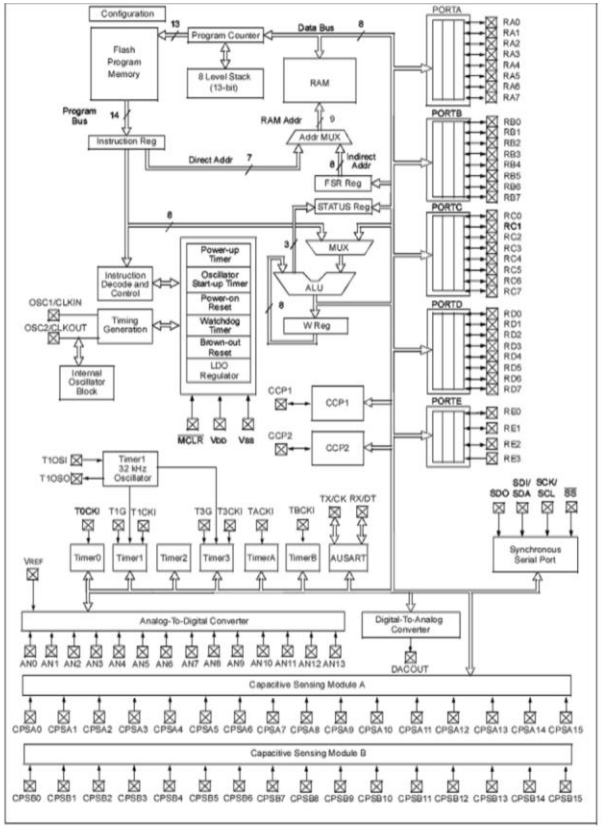


Figura 5.6: Organizzazione interna di un microcontrollore Microchip PIC© della serie 16(L)F707. Il chip segue la filosofia Harvard con bus e memorie separati per istruzioni e operandi.

Example

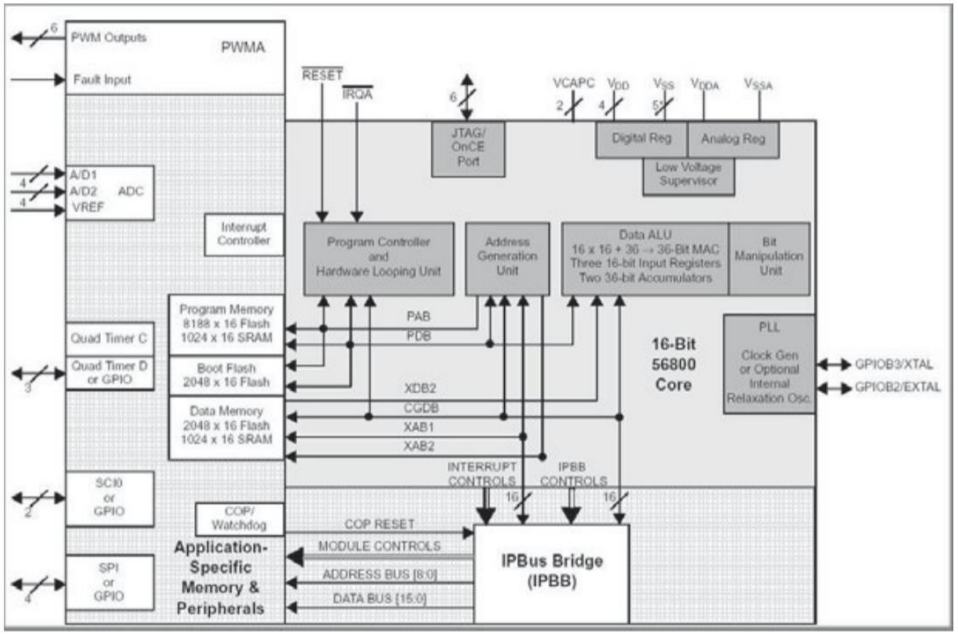


Figura 5.7: Organizzazione interna di un Digital Signal Controller di produzione NXP, appartenente alla serie 56F8XXX (ex Freescale, ex Motorola). Si noti la tipica suddivisione delle memorie e dei bus, secondo l'impostazione di Harvard.

Example

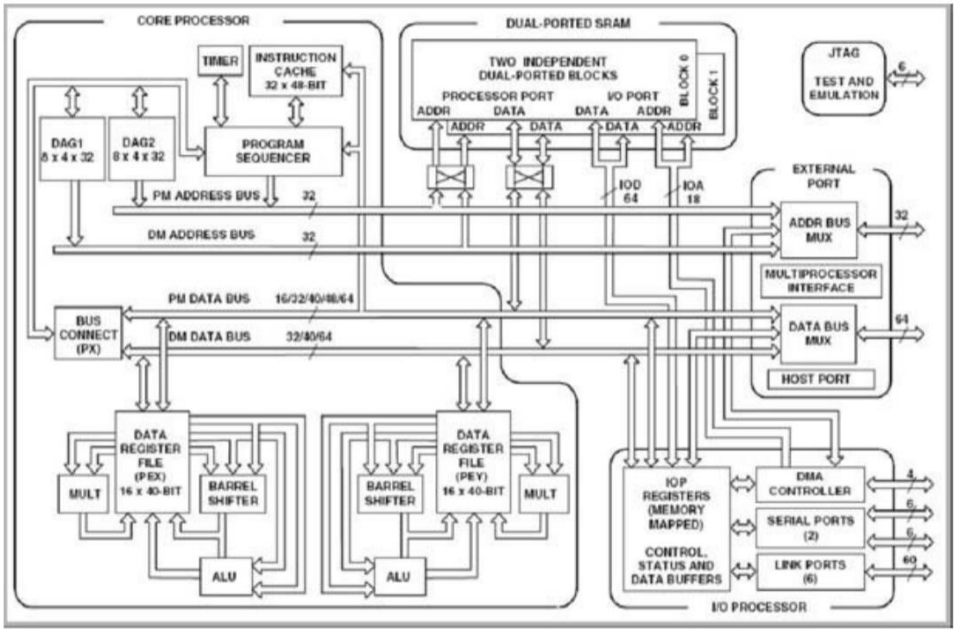


Figura 5.8: Organizzazione interna di un DSP Analog Devices della serie 21XXX. Si noti l'uso di due memorie separate, entrambe di tipo "dual port".

Example

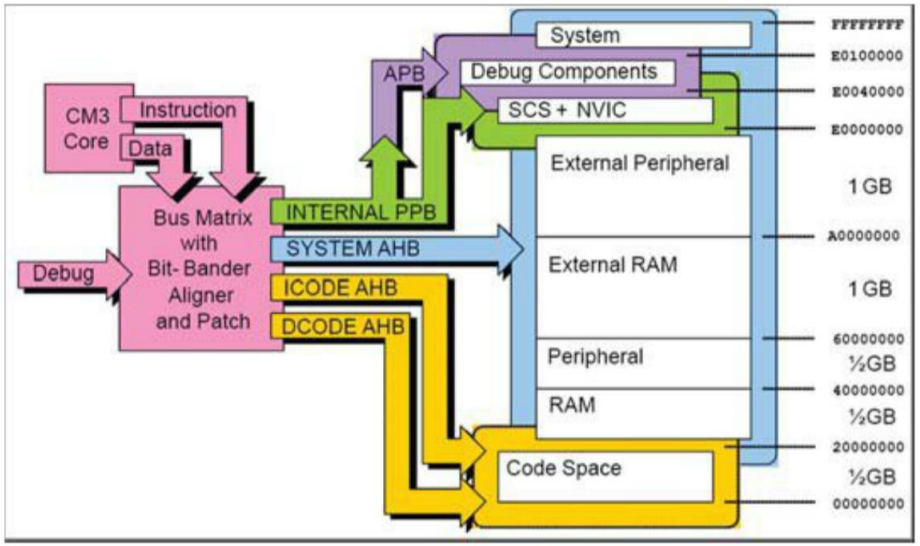


Figura 5.9: Organizzazione interna di una CPU della serie ARM Cortex M3©. Sono presenti *numerosi* bus separati, secondo lo standard AMBA.

User architecture

- A *processor architecture* can be defined as the combination of resources that allow programming the processor in *assembly* language. This is also called *user architecture.*
- Independently of its implementation, the task of an architecture is that of providing an efficient support for the processor programming.
- Relevant aspects of the processor architecture are:
  - Instruction structure,
  - Data addressing modes,
  - Functions implemented by the instructions.

Architecture characteristics

- The most interesting characteristics of an instruction sets are:
  - Symmetry,
  - Orthogonality,
  - Regularity,
  - Compactness,
  - Speed of execution,
  - Debugging simplicity.

- *Symmetry* when instructions do not behave differently depending on operands. (example: when some registers can be copied to memory and others not).
- *Orthogonality* refers to the uniformity of addressing modes to the different instructions: we have it if all addressing modes are applicable to all instructions.

Architecture characteristics

- *Regularity:* when the instruction words (IWs) have the same length and the size of fields dedicated to specific functions (e.g., *OpCode,* registers) are always the same.
- *Compactness* of the instruction set refers to the size of the machine code generated by the assembler for a certain program.
- The *Execution speed* depends on the organization at circuit level of the processor. Architectures symmetric, orthogonal, and regular simplify decoding and execution, reducing the complexity of the overall circuit.
- *Debugging simplicity* means readability of the code generated by the compiler or programmer. It is important for code reuse and portability.

RISC and CISC architectures

- A possible classification of architectures is the following:
  - RISC (reduced instruction set computer) architectures
    - Registers based
    - Accumulator/stack based (obsolete).
  - CISC (complex instruction set computer) architectures
    - Extended von Neumann
    - High level language oriented.

RISC architectures

- Instructions are less numerous (few dozens) and as simple as possible.
- Simple both in terms of instruction *format* and instruction *function.*
- Since instruction decoding (almost always *cabled*) is very simple, RISC architecture allow instruction execution in a single machine cycle.
- They allow a high reduction in silicon area need, and thus a low final cost.

- Nowadays, CPU organization is almost always of the kind “with many registers”.
- Historically, first implementations of RISC processors had an organization based on a single register (called accumulator) and a small stack. Thus, all instructions were without operands.

CISC architectures

- Are characterized by very numerous instruction sets (hundreds of instructions), with variable structure, also very complex.
- Decoding in this case is microprogrammed.
- Execution requires different cycle times.
- CISC architectures were very common in GPPs, but rare in µCs and DSPs.
- Many CISC variants:
  - Extended von Neumann organizations, empowered with cache memories, coprocessors, debugging and program tracing circuits …
  - High-level language oriented to facilitate compilation.

µC and DSP architectures

- Some old µC architectures (of the 70s but still in use) were CISC (e.g., Intel 8XC196, Reneas H8S). Current µC architectures are RISC.
- DSPs are mainly RISC but their architecture have evolved in time for providing always higher performances.
- *Instruction words* have become more and more complex, split in several fields to improve flexibility, but often loosing symmetry and orthogonality.
- Another trend has been the *Very Long Instruction Word (VLIW).* In this architecture is in possible to combine in a single macro-instruction several (2, 4, 8) small instructions, which are executed in parallel, often in a single cycle, using several ALUs. Only arithmetic instructions are used in VLIW. Thus, we have IWs of different sizes.
- In *Single Instruction Multiple Data* (SIMD) architecture, the same instruction is executed on different operands, allowing parallelism on data.

µC and DSP instruction sets

- DSPs always include a multiplication instruction performed in a single cycle (and many include also a division). Multiplication can be combined with shift operations and addition with an accumulator in the same cycle.
- Most DSPs allow to control the result of the operations and perform truncation or rounding. It is also possible to work with a saturated arithmetic.
- Moreover, they allow
  - data moving, data comparison, bit manipulation,
  - code repetition, jumps, branches and subroutine calls,
  - floating point emulation.
- Many current µCs have an HW organization identical to DSPs. They have a similar instruction set, RISC based, and even HW multipliers. But typically they have a lower parallelism to reduce costs.

µC and DSP instruction sets

- DSPs and µCs differ from GPPs for data addressing. Common to all are:
  - *Immediate* addressing, with operand in IW
  - *Register* addressing, operand in a register specified in IW
  - *Direct or absolute* addressing, IW has the address of memory operand
  - *Indirect* addressing, IW has memory address of operand address (pointer)
- More specific of DSPs and µCs:
  - *Indirect addressing with auto-increment*, where IW has the address to a pointer, to which an increment is automatically added.
  - *Indirect addressing with off-set*, where an offset included in the IW is added to the pointer.
- DSPs and µCs often have separate *Address Generation Units (AGU)* to allow:
  - Indirect addressing from register; Circular or modulo addressing;
  - Bit reversed addressing

Modulo addressing

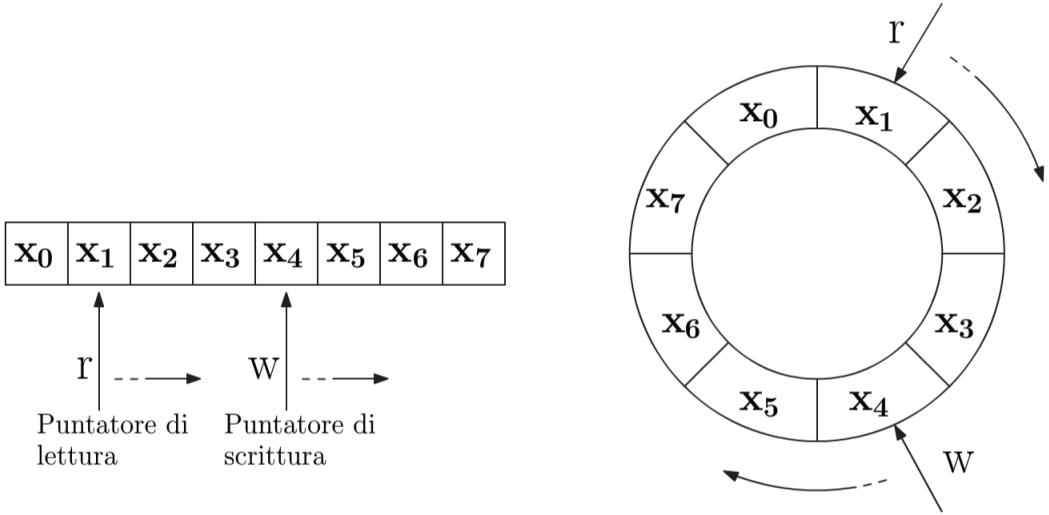


Figura 5.11: Indirizzamento circolare o "in modulo": l'hardware è predisposto in modo da consentire che, una volta raggiunta l'estremità di una lista, un puntatore di lettura o scrittura venga automaticamente re-impostato al valore corrispondente all'estremità opposta. Si implementa così una struttura dati circolare, in cui il successore di un estremo, nella direzione opportuna, è l'estremo opposto.

Bit Reversal addressing

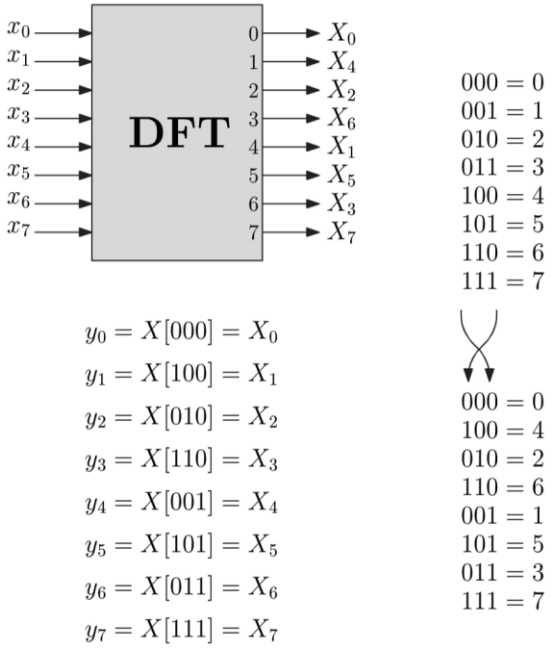


Figura 5.12: Indirizzamento a bit rovesciati: gli algoritmi DFT ottimizzati producono un vettore della trasformata X che risulta non ordinato. L'ordine corretto si può però facilmente ripristinare leggendo gli indirizzi delle componenti del vettore della trasformata a bit rovesciati.

Control strategies

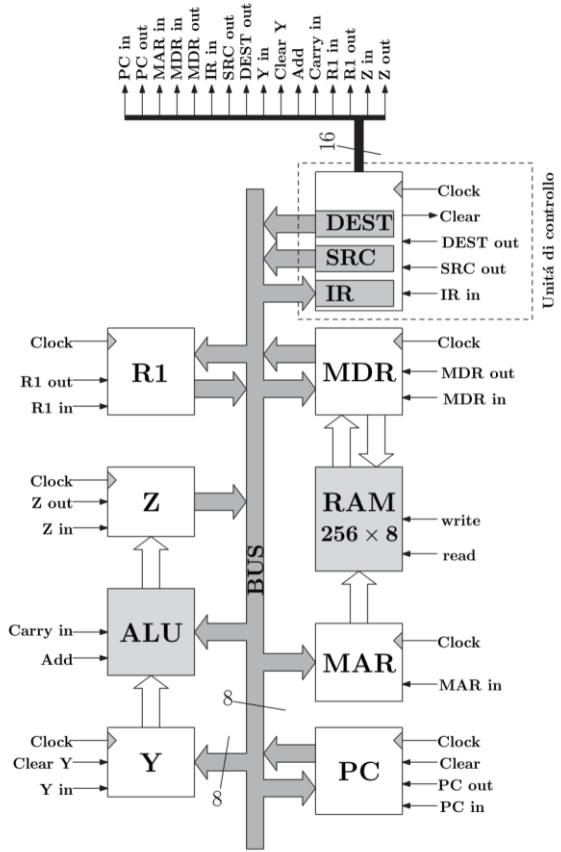


Figura 5.13: Un ipotetico processore a 8 bit.

Control strategy

- Let us consider the hypothetical processor of the previous slide.
- The control unit decode and execute the instructions, and update the *program counter* (PC), fetching the next instruction.
- The control unit is here composed of three register:
  - *IR* has the *OpCode* of the instruction
  - *SRC* can contain a parameter included in the IW or a memory address, often expressed in *relative* terms as a PC increment.
  - *DEST* can contain an address or the pointer to a register where to write the result.
- From the analysis of the OpCode, the control unit must provide the sequence of control signals necessary for
  1. Fetch and update of the PC
  2. Execution of the current instruction.

Control strategy

- Historically, two possible approaches have been followed for the control unit:
  - *Microprogrammed* approach
  - *Cabled* approach
  - Microcontrollers with old architectures are microprogrammed. More recent ones, especially RISC ones, are cabled.
- Two possible *clocking* strategies:
  - *Multi-cycle control*
    - Fetch, decode, execute performed with multiple clock period.
  - *Single-cycle control*
    - Fetch, decode, execute performed in a single period.
  - Single-cycle control strategy is employed only in cabled controls.
  - Multi-cycle control strategy is used both in all microprogrammed controls and in some cabled controls.

Microprogrammed control

- Is implemented with a control unit that replicate the structure of a simple CPU with a memory, a PC, an ALU, called *microcode engine*.
- Each macro-instruction corresponds to a microcode, composed by some words. Microcodes are stored in a ROM memory.
- Two possibilities:
  - *Horizontal microprogramming*: the control unit execute the code strictly sequentially, starting from the address pointed by the OpCode.
  - *Vertical microprogramming:* jumps are possible and allow to repeat microcode segments, i.e., the introduction micro-subroutines.
- The *microcode* is composed of words, whose bits directly assert/negate specific control signals.
- The microcode *wordlenght* depends on the number of control signals.

Microprogrammed control

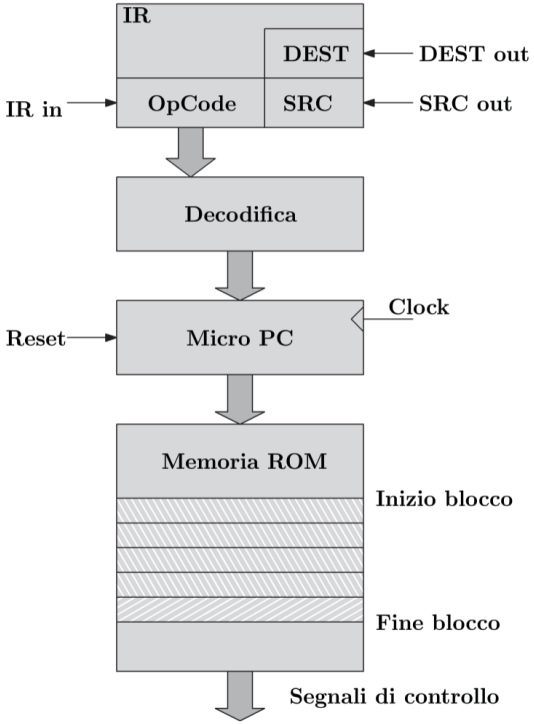


Figura 5.14: Schema a blocchi semplificato di una unità di controllo microprogrammata.

Cabled control

- The microcode engine is replaced by a *combinatorial logic circuit* that generates directly the control signals form the OpCode of the current instruction, managing also temporizations.
- Includes a *secondary clock generator*, whose purpose is the time distribution of control signal activations.
- Let us assume that a single instruction is executed in 7 periods.
- The secondary clock generator is a clock divider by 7 that generates 1 pulse every 7 clock periods, and feeds a shift register of 6 FlipFlops.
- The decoder activates one output line for each OpCode.
- The combinatorial network, composed by AND and OR gates, feeds the control lines on the basis of the OpCode and of secondary clock state (from 1 to 7).
- The solution provides *very fast response*, with *little silicon area occupation,* but *lack flexibility*, and could require *nop* cycles to manage shorter instructions

Cabled control

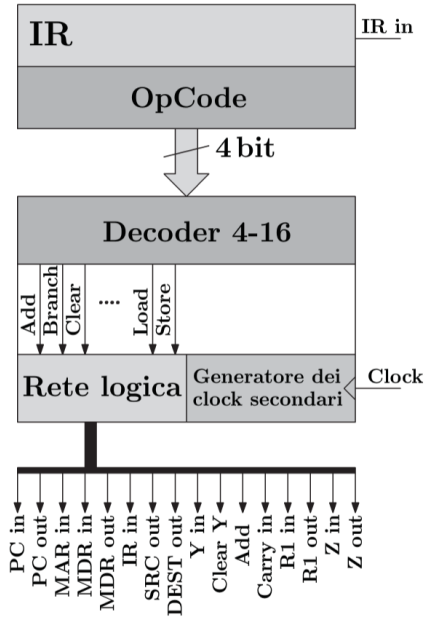


Figura 5.15: Unità di controllo cablata: è visibile un decodificatore delle istruzioni (a 4 bit), un generatore dei clock secondari e una rete logica combinatoria che asserisce i segnali di controllo del processore, elaborando sia i clock secondari che le linee del bus gestito dal decodificatore.

Cabled control

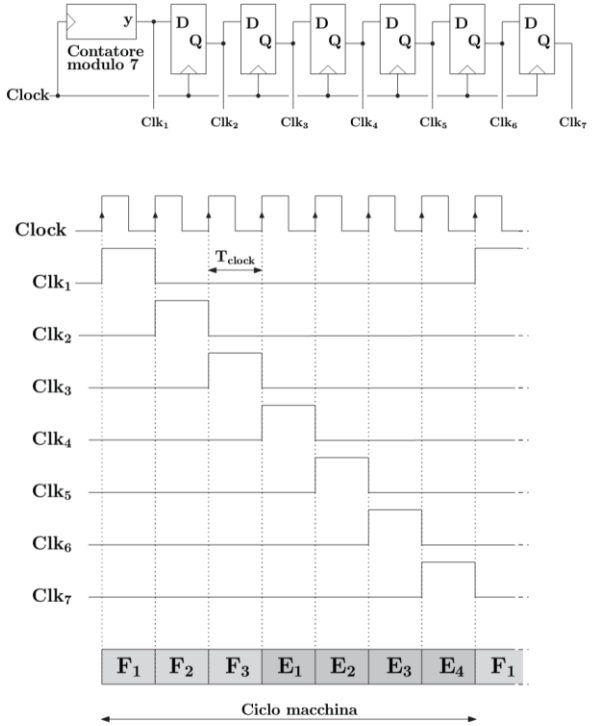


Figura 5.16: Circuito per la generazione dei segnali di clock secondari richiesti da una unità di controllo cablata.

Example

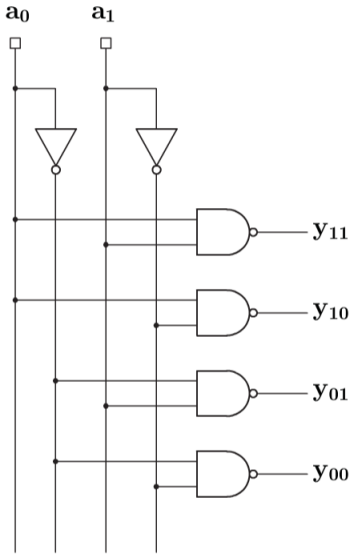


Figura 5.17: Schema di principio di un decodificatore binario, o demultiplexer, a 2 bit. Ciascuna delle 4 combinazioni di ingresso attiva, portandola, in questo caso, a livello logico *basso*, una e una sola delle 4 uscite.

Cabled control

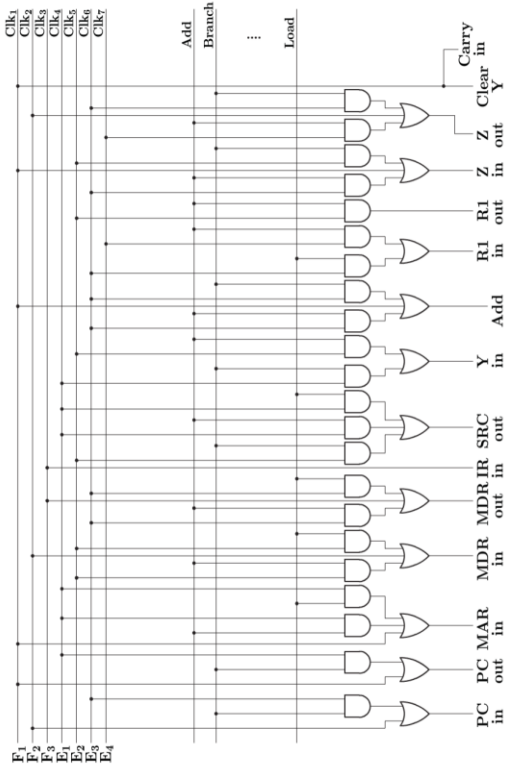


Figura 5.18: Una parte della rete combinatoria richiesta dal controllo del processore di Fig. 5.13.

Single cycle organization

- Only the availability of multiple resources allow a single cycle temporization.
- It requires at least that the fetch phase is performed simultaneously to decode and execute.
- It imposes the following system requirements:
  - Separate data memory and instruction memory;
  - Separate ALU for PC increment;
  - Flexible PC increment for managing jumps without main ALU intervention.
- Unless the instruction set is very simple, the single cycle organization is often *inefficient*. It is the most onerous instruction that determine the clock period.
- On the contrary, in multi-cycle organization, it is the slowest functional unit (ALU or memory) that determines the minimum period.
- It is possible to combine advantages of both, using a *pipeline* organization.

See:

- Simone Buso, «Introduzione alle applicazioni industriali di Microcontrollori e DSP» Società editrice Esculapio, 2018
  - Chapter 5