centro appunti
www.centroappunti.it
Corso Luigi Einaudi, 55 - Torino

Appunti universitari
Tesi di laurea
Cartoleria e cancelleria
Stampa file e fotocopie
Print on demand
Rilegature

NUMERO: 1566A - ANNO: 2015

A P P U N T I

STUDENTE: Fissore

MATERIA: Elettronica dei Sistemi Digitali SLIDE. Prof.Zamboni

Il presente lavoro nasce dall'impegno dell'autore ed è distribuito in accordo con il Centro Appunti.
Tutti i diritti sono riservati. È vietata qualsiasi riproduzione, copia totale o parziale, dei contenuti inseriti nel presente volume, ivi inclusa la memorizzazione, rielaborazione, diffusione o distribuzione dei contenuti stessi mediante qualunque supporto magnetico o cartaceo, piattaforma tecnologica o rete telematica, senza previa autorizzazione scritta dell'autore.

ATTENZIONE: QUESTI APPUNTI SONO FATTI DA STUDENTI E NON SONO STATI VISIONATI DAL DOCENTE.
IL NOME DEL PROFESSORE, SERVE SOLO PER IDENTIFICARE IL CORSO.

MicroController Concepts

Appunti presi a partire dalle video-lezioni di Elettronica dei Sistemi Digitali, aa. 2013/2014.

Saranno probabilmente disponibili appunti simili anche per i corsi del quarto anno di Microelettronica Digitale e Sistemi Elettronici a Basso Consumo.

Slide modificate da Giorgio Fissore
Disponibili in centro stampa

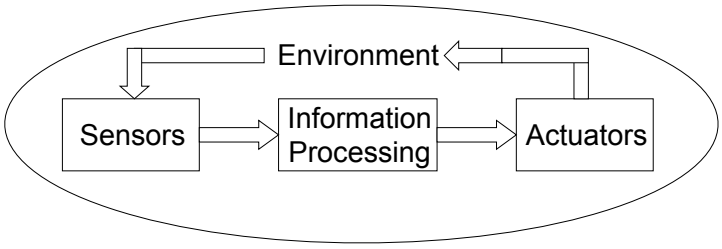
Nella prima metà del corso abbiamo imparato a progettare l'hw, sia nella sua unità di esecuzione, il datapath, che nella sua unità di controllo.
L'elettronica digitale, però, prevede anche la possibilità di utilizzare macchine come quelle che abbiamo imparato a progettare, ma progettate da altri, con cui eseguire una grandissima varietà di

Microcontroller systems

- Nearly any computing system other than a general purpose computer
- Microcontroller systems are often called embedded systems
- A computing system, embedded within a larger system, and repeatedly performing a specific task *("the application")*

Un sistema a microcontrollore (uC) si differenzia nettamente da un microprocessore (uP), e viene oggi chiamato sistema embedded: un sistema integrato in un certo ambiente che riceve dei segnali dall'esterno ed esegue un algoritmo su questi dati che gli arrivano; va poi a produrre degli effetti esterni.

Embedded Systems

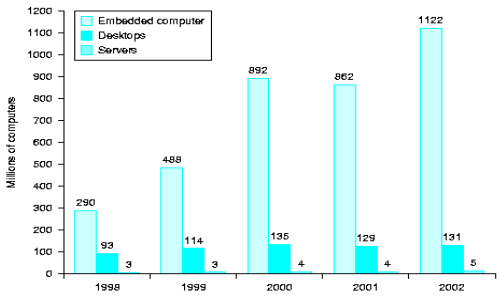


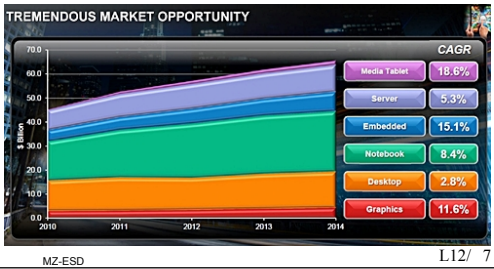
- Environment to Environment
- Sensors + Information Processing + Actuators

Mod by Giorgio Fissore

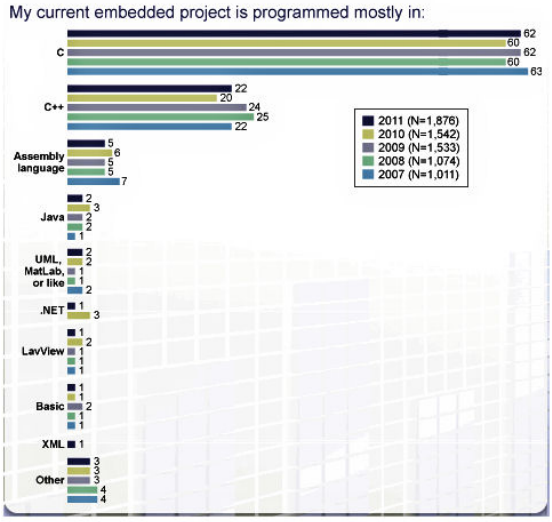
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 3 di 334

The Computer Market





ES: Programming languages used



La maggior parte dei sistemi embedded sono
-programmati in C (~60% delle applicazioni)
-programmati in C++ (~24%)
>>> l'84% dei sistemi embedded viene programmato in C/C++
>>> appena il 5% viene programmato in assembler (e probabilmente da vecchi elettronici)
>>> nessuno si sognerebbe invece di programmare un embedded in java: questo linguaggio di programmazione infatti è progettato per girare dappertutto, ma non è assolutamente ottimizzato per niente!!
>>> il motivo per cui si progetta in C/C++ è che questi due linguaggi vanno praticamente alla stessa velocità dell'assembler!!
>> la filosofia di progetto è: fai tutto in C; poi se c'è qualche punto particolarmente critico, prova a limare con l'assembler

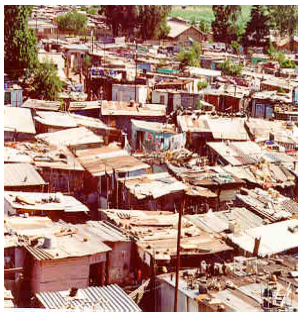
Moore's Law

- In 1965 Intel co-founder Gordon Moore predicted that IC transistor capacity would double every 18 months

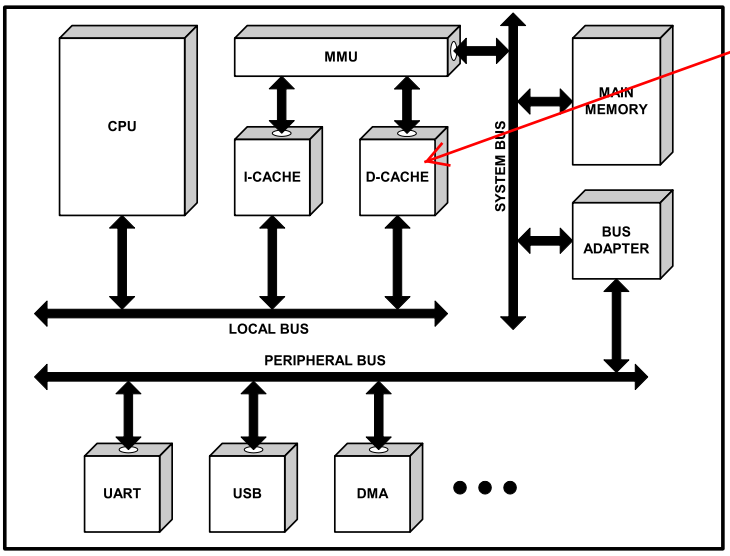
Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 5 di 334

Sub-optimal vs. optimal Architecture



Typical Embedded System

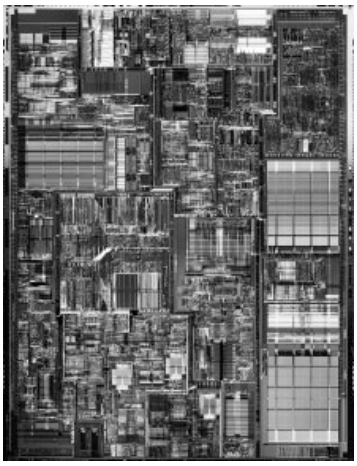


La cache non è presente in molti uC

Caratteristica tipica dei uC (dei sistemi embedded), che lo differenzia nettamente da un uP, è la presenza di svariate

What is a Microprocessor ?

Pentium 4 microphotograph



All building units of the processor are integrated on a single chip while no peripheral block is integrated inside!!

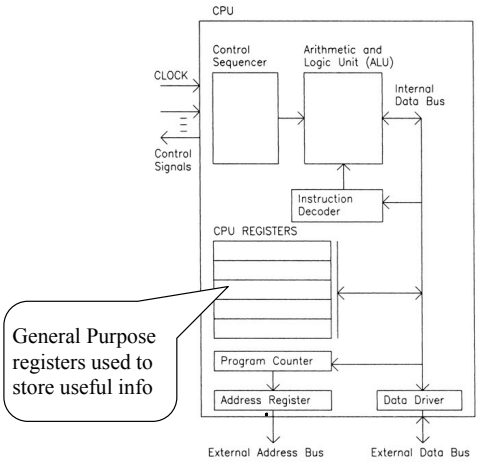
Se si prende un uP così com'è, questo non è in grado di fare assolutamente nulla, ma bisogna costruirgli un sistema intorno.
Se si guarda un uC invece è presente la memoria, gli I/O...

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 7 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 334

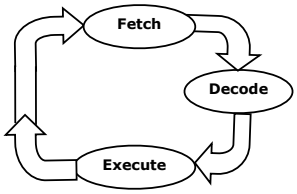
Inside the Microprocessor



- The CPU executed the program instructions
- ALU performs arithmetic and logic operations
- The ID tells the ALU what to do with the data
- PC is a special register that tells the CPU where to get the next instruction
- PC, AR and DR are special registers

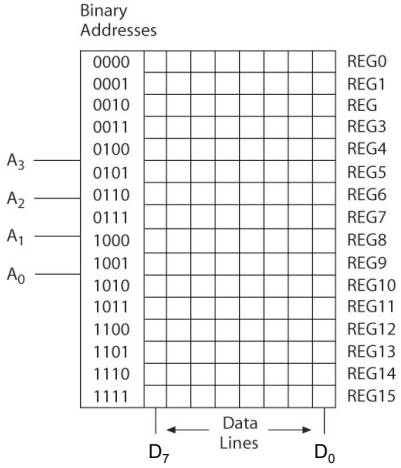
Microprocessor Basic Operation

- Program (instructions) and Data are stored in Memory
- Each instruction is read (fetched) from memory, interpreted (decoded), and executed
  - Arithmetic Logic Unit (ALU) performs operations on data
  - Data is transferred (register, memory, I/O)
- Program Counter (PC) indicates current location of program in Memory and is automatically incremented after each instruction
- Each instruction can take several clock cycles



Il funzionamento come già visto avviene attraverso le tre fasi in successione di:
-fetch, cerco cosa devo fare
-decode: capisco cosa devo fare
-execute: eseguo l'operazione.
E' questo il flusso base di operazioni di un uP.

Memory



- Storage Device
  - Addresses
  - Registers
- Major Categories
  - Read/Write Memory (R/W)
  - Read-only-Memory (ROM)

Mod by Giorgio Fissore

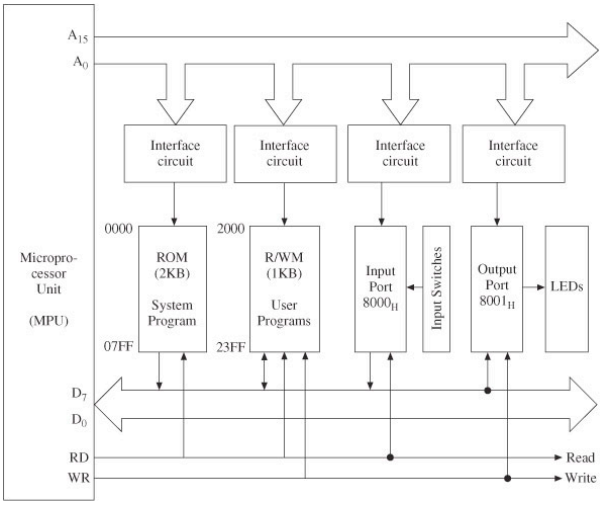
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 9 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 334

Microprocessor Architecture

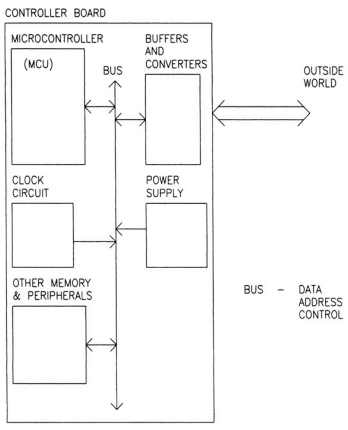
- The MPU communicates with Memory and I/O using the System Bus
  - Address bus
    - Unidirectional
    - Memory and I/O Addresses
  - Data bus
    - Bidirectional
    - Transfers Binary Data and Instructions
  - Control lines
    - Read and Write timing signals

Example Microprocessor System



What is a Microcomputer System ?

- It is a computing system based on microcontroller.



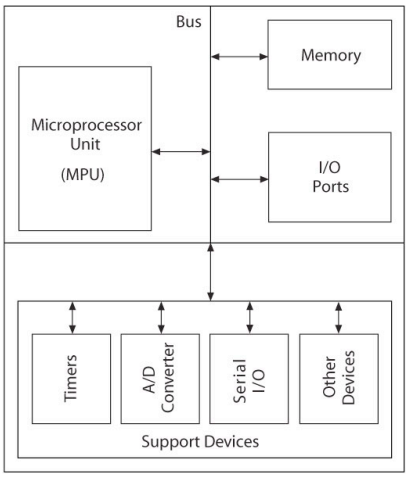
Un uC, invece ha qualcosa in più.
Ha al suo interno un processore, la CPU, ma oltre ciò, è anche presente tutta la parte di memoria (rom e ram); il bus di sistema è interno all'integrato. Vi è anche in esso tutta la gestione del clk, tutta la gestione delle periferiche, la gestione dei buffer e dei livelli elettrici per comunicare con l'esterno..
>> tutti i circuiti che servono per gestire le periferiche sono embedded dentro il sistema
>> un uC è un sistema single chip >> con un solo integrato posso gestire tutto il sistema.
(buffer e convertitori, gestione dei bus, generazione del clk, gestione della potenza)

Mod by Giorgio Fissore

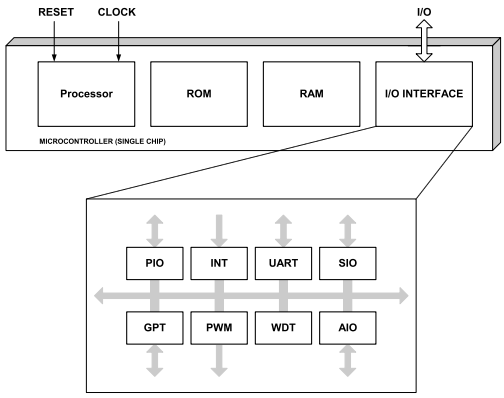
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 11 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 334

Block Diagram



Microcontroller Organization



Microcontroller Functional Units

- **CPU**: Central Processing Unit (4,8,16,32 bit data bus)
- **ROM**: Read Only Memory (Firmware)
- **RAM**: Random Access Memory (Register File, Processor Stack, Temporary data)
- **PIO**: Parallel I/O (relays, sensors)
- **INT**: Interrupt Inputs (external/internal sources)
- **UART**: Universal Asynchronous Receiver Transmitter (e.g. RS232)
- **GPT**: General Purpose Timer (optional event counter)
- **PWM**: Pulse Width Modulator (motor controller)
- **WDT**: Watch Dog Timer (automatic reset)
- **AIO**: Analog I/O (ADC & DAC)

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 13 di 334

The Bus

- The bus provides the communication infrastructure among the various components of the system
- **Data bus** carries the information being transmitted/received.
- **Address bus** tells where the information is being transferred to/from.
- **Control bus** specifies when the information transfer take place by coordinating the access to the data bus and the address bus, and directs the data from/to the specific components.

Control bus: viaggiano su di esso l'insieme di segnali di controllo necessari per il trasferimento delle informazioni (es dicono se lettura/scrittura, se il dispositivo è pronto o meno per ricevere dati,...)

Computer Architectures (complementary stuff)

- Princeton (Von Neumann) versus Harvard Architecture
- CISC versus RISC processors
- Microprocessors and Microcontrollers

Computer Concepts

- Computer:
  - Hardware:
    - Processor: “brain”, CPU
      - Datapath: registers and ALU
      - Control unit: hardware instrucion logic.
    - Memory: place to store software programs and data
    - I/O devices: enter data/programs into the computer/display outputs
  - Software: programs
    - A program is a set of instructions that the computer hardware can execute.

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 15 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 334

High-level languages

- Syntax of a high-level language is similar to English, (C/C++, Java, Fortran, PASCAL…)
- A translator is required to translate the program written in a high-level language -- done by a compiler
- High-level languages allow the user to work on the program logic
  at higher level.

Source Code

A program written in assembly or high-level language

Object Code

The output of an assembler or compiler

Software

- Machine Language
  - Binary Instructions
  - Difficult to decipher and write
    - Error-prone
  - All programs converted into machine language for execution

Instruction	Hex	Mnemonic	Description	Processor
10000000	80	ADD B	Add reg B to Acc	Intel 8085
00101000	28	ADD A, R0	Add Reg R0 to Acc	Intel 8051
00011011	1B	ABA	Add Acc A and B	Motorola 6811

Software

- Assembly Language
  - Machine instructions represented in mnemonics
  - One-to-one correspondence
  - Efficient execution and use of memory
  - Machine-specific

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 17 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 334

Data Format (8-bit)

- Binary Coded Decimal Numbers (BCD)
  - 8 bits of a number divided into groups of four
    - Each group represents a decimal digit from 0 to 9
  - A_H through F_H are invalid
  - Example: $0010\ 0101_{BCD} = 25_{10}$

Data Format (8-bit)

- American Standard Code for Information Interchange (ASCII)
  - Seven-bit alphanumeric code with 128 combinations (00_H to $7F_H$)
  - Represents English alphabet, decimal digits from 0 to 9, symbols, and commands

MPU-Based Systems

- System hardware
  - Discrete components
    - Microprocessor, Memory, and I/O
  - Components connected by buses
    - Address, Data, and Control
- System software
  - A group of programs that monitors the functions of the entire system

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 19 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 334

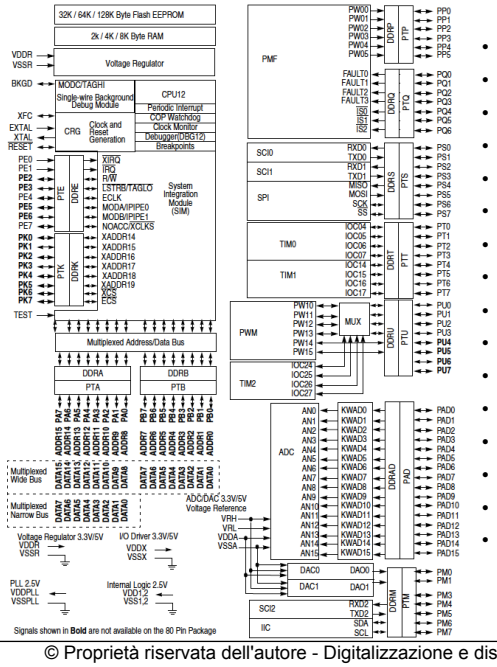
9S12 Hardware Overview, Subsystems, and Memory System

Slide modificate da Giorgio Fissore
Disponibili in centro stampa

Overview

- 9S12 hardware overview
- Subsystems
- Memory System

MC9S12E128A Block Diagram

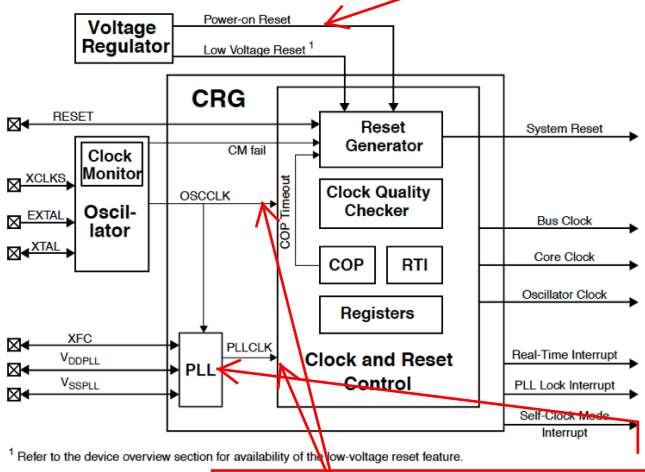


- Processor CPU12
- xx Kbytes Flash EEPROM
- yy Kbytes RAM
- zz bytes EEPROM
- PAD A/D 16 channels 10 bits
- PTT Timer
- PTS Serial SCI, SPI, 2 I/O
- PTU 4 PWM, 4 I/O
- PTA Address/Data
- PTB Address/Data
- PTE Control
- PTK Extended Address
- BDI Debug

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 21 di 334

Clock and Reset Generation Block (CRG)



1/10/2012

Nel caso in cui si abbassasse la tensione devo resettare in maniera particolare.

Blocco per la generazione di CLK e Reset estremamente complesso.

5 modi diversi per resettare, tra gli altri, un reset per lavorare anche se il clk si inchioda, con un clk = 1 MHz

3 clk diversi!!!

Moltiplicatore, o divisore, di frequenza; impiega però parecchio tempo per agganciarsi alla frequenza

possiamo prendere il clk da qui o dal pll

Clock and Reset Generation Block (CRG)

Features (from data sheets)

- **Phase-locked loop (PLL) frequency multiplier**
  - — Reference divider
  - — Automatic bandwidth control mode for low-jitter operation
  - — Automatic frequency lock detector
  - — CPU interrupt on entry or exit from locked condition
  - — Self-clock mode in absence of reference clock
- **System clock generator**
  - — Clock quality check
  - — Clock switch for either oscillator- or PLL-based system clocks
  - — User selectable disabling of clocks during wait mode for reduced power consumption
- **Computer operating properly (COP) watchdog timer with time-out clear window**
- **System reset generation from the following possible sources:**
  - — Power-on reset
  - — Low voltage reset (when available)
  - — COP reset
  - — Loss of clock reset
  - — External pin reset
- **Real-time interrupt (RTI)**

1/10/2012 MZ-ESD L13/ 8

Clock and Reset Generation Block (CRG)

- CRG generates the clock signals required by the instruction execution and all peripheral operations.
- The clock signal has the form of square waveform.
- Crystal oscillators are often used to generate clock signals.
- The crystal oscillator output is sinusoidal wave that is squared up internally before it can be used.
- The CRG block also has a PLL circuit that can multiply the frequency of the incoming clock signal.
- The CRG can also accept oscillator output (square waveform) directly.
- The XCLKS signal must be tied low (for MC9S12DP256B) in order to use external clock signal.

Mod by Giorgio Fissore

1/10/2012 MZ-ESD L13/ 9

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 334

Phase Locked Loop (PLL)

- The frequency of the PLLCLK is controlled by registers SYNR and REFDY using the following equation:

$$PLLCLK = 2 \times OSCCLK \times \frac{(SYNR + 1)}{(REFDV + 1)}$$

	7	6	5	4	3	2	1	0
	0	0	SYN5	SYN4	SYN3	SYN2	SYN1	SYN0
reset:	0	0	0	0	0	0	0	0

The CRG synthesizer register (SYNR)

	7	6	5	4	3	2	1	0
	0	0	0	0	REFDV3	REFDV2	REFDV1	REFDV0
reset:	0	0	0	0	0	0	0	0

The CRG reference divider register (REFDV)

In un PLL il clk si genera partendo da un clk esterno, moltiplicato per il rapporti tra due numeri scritti in due registri.
>> per far funzionare il blocco del PLL devo prima programmarlo.

Phase Locked Loop (PLL)

	7	6	5	4	3	2	1	0
	PLLSEL	PSTP	SYSWAI	ROAWAI	PLLWAI	CWAI	RTIWAI	COPWAI
reset:	0	0	0	0	0	0	0	0

PLLSEL: PLL select bit
0 = system clocks are derived from OSCCLK
1 = system clocks are derived from PLLCLK
PSTP: pseudo stop bit
This bit controls the functionality of the oscillator during the stop mode.
0 = oscillator is disabled in stop mode
1 = oscillator continues to run in stop mode (pseudo mode). The oscillator amplitude is reduced.
SYSWAI: system clocks stop in wait mode bit
0 = The system clocks continue to run in wait mode.
1 = The system clocks stop.
ROAWAI: Reduced oscillator amplitude in wait mode bit
0 = Normal oscillator amplitude in wait mode
1 = Reduced oscillator amplitude in wait mode
PLLWAI: PLL stops in wait mode bit
0 = PLL keeps running in wait mode
1 = PLL stops in wait mode. The CRG will clear the PLLSEL bit before entering wait mode. The PLLON bit remains set during wait mode but the PLL is powered down.
CWAI: core stops in wait mode bit
0 = Core clock keeps running in wait mode.
1 = Core clock stops in wait mode.
RTIWAI: RTI stops in wait mode bit
0 = RTI keeps running in wait mode.
1 = RTI stops and initializes the RTI dividers whenever the part goes into wait mode.
COPWAI: COP stops in wait mode bit
0 = COP keeps running in wait mode
1 = COP stops and initializes the COP dividers whenever the part goes into wait mode.

The CRG clock select register (CRGSEL)

Selection of PLL for **clock generation** is controlled by the CRGSEL register.

Phase Locked Loop (PLL)

	7	6	5	4	3	2	1	0
	CME	PLLON	AUTO	ACQ	0	PRE	PCE	SCME
reset:	0	0	0	0	0	0	0	0

CME: clock monitor enable bit
0 = clock monitor is disabled
1 = clock monitor is enabled. Slow or stopped clocks will cause a clock monitor reset sequence or self clock mode
PLLON: phase lock loop on bit
0 = PLL is turned off
1 = PLL is turned on. If AUTO bit is set, the PLL will lock automatically.
AUTO: automatic bandwidth control bit
0 = automatic mode control is disabled and the PLL is under software control, using ACQ bit.
1 = high bandwidth filter is selected
ACQ: acquisition bit (if AUTO bit = 1, this bit has no effect)
0 = low bandwidth filter is selected
1 = high bandwidth filter is selected
PRE: RTI enable during pseudo stop bit
0 = RTI stops running during pseudo stop mode
1 = RTI continues running during pseudo stop mode
PCE: COP enable during pseudo stop bit
0 = COP stops running during pseudo stop mode
1 = COP continues running during pseudo stop mode.
SCME: self clock mode enable bit
0 = detection of crystal clock failure causes clock monitor reset
1 = detection of crystal clock failure forces the MCU in self clock mode

The CRG PLL control register (PLLCTL)

PLL circuit is also controlled by the **PLLCTL** register.

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 25 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 334

Hardware Overview

- Timing System
- Memory System
- Interrupt and Reset System
- Serial Communication System
- Port System
- Data Conversion System

Hardware Overview (cont)

In base alle versioni posso scegliere il uC anche con diversi tagli di memoria

- Low power consumption - CMOS technology,
- 16-bit data path (it can be used in 8-bit mode)
- Different memory configurations:
  - 1..12 K-byte RAM
  - 0.5...4 K-byte EEPROM
  - 16..240 K-byte FLASH
- 8-channel, 16-bit timer, configure each channel separately as input or output
- 16-bit pulse accumulator

qui posso modificare anche la singola locazione

Hardware Overview (cont)

- Real-time interrupt capability - accomplish operations on recurring basis - reminder
- Serial communications:
  - Serial Communication Interface (SCI)
    - asynchronous, 2 channels
  - Serial Peripheral Interface (SPI)
    - synchronous
- 8-channel, 8-bit analog-to-digital converter

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 27 di 334

Subsystems Port System

- Port related registers:
  - Data Direction Register (DDRx): configures Port as input/output (1: output, 0: input)
  - Pull Up Control Register (PUCR): provides built-in pull-up resistor for interface applications
  - Reduced Drive of I/O Lines Register (RDRIV): reduces current drive capability of pin
  - Port E Assignment Register (PEAR):
    - provides alternate bus functions in expanded mode
    - signals used in memory expansion applications

Ogni porta ha per esempio un:
-DDR: per decidere I/O
-PUCR: decide se attivare Rpu
-RDRIV: decide se lavorare a bassa corrente.
>>> differenza rispetto a uP
>>> posso programmare anche i livelli elettrici del uC, e la gestione degli I/O

Registers I/O (some examples...)

Table 2-19. Block Register Map (G1)

Global Address Register Name		Bit 7	6	5	4	3	2	1	Bit 0
0x0000 PORTA	R	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
	W								
0x0001 PORTB	R	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
	W								
0x0002 DDRA	R	DDRA7	DDRA6	DDRA5	DDRA4	DDRA3	DDRA2	DDRA1	DDRA0
	W								
0x0003 DDRB	R	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
	W								
0x0004 PORTC	R	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
	W								
0x0005 PORTD	R	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0
	W								
0x0006 DDRC	R	DDRC7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
	W								
0x0007 DDRD	R	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
	W								
0x0008 PORTE	R	0	0	0	0	0	0	PE1	PE0
	W								
0x0009 DDRE	R	0	0	0	0	0	0	DDRE1	DDRE0
	W								
0x000A–0x000B Non-PIM Address Range	R	Non-PIM Address Range							
	W								
0x000C PUCR	R	0	BKPUE	0	PDPEE	PUPDE	PUPCE	PUPBE	PUPAE
	W								
0x000D Reserved	R	0	0	0	0	0	0	0	0
	W								

The Timing System - Standard Timer Module (TIM)

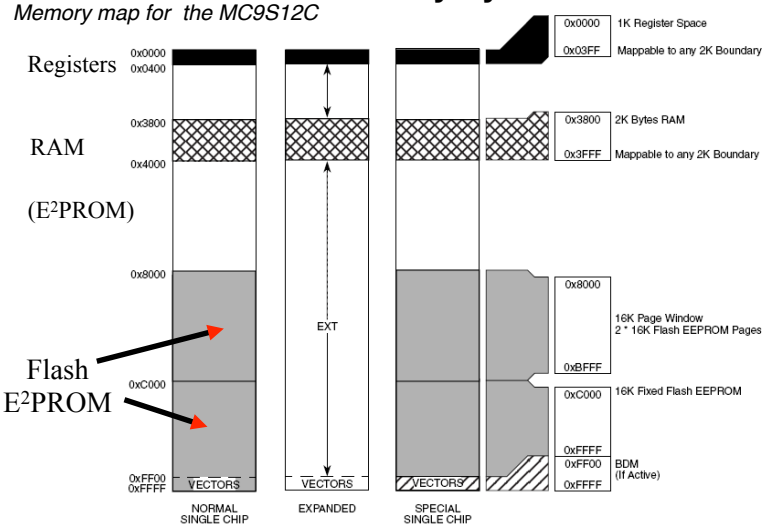
- TIM contains 16-bit programmable counter
- Provides following precision timer functions:
  - Output compare: generate precision output signals such as periodic digital waveforms, pulses, etc.
  - Input capture: measure the characteristics of incoming signals such as frequency, period, duty cycle, pulse length
  - Pulse accumulator: count external events
  - Pulse Width Modulated (PWM) signal generation
    - PWM not available in some S12 variant

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 29 di 334

9S12 Subsystems
The Memory System

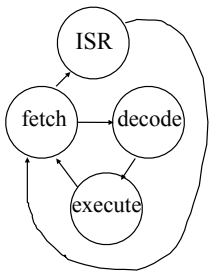
Memory map for the MC9S12C



Siccome la mia macchina è a 16 bit, può programmare fino a 2'16 bit >> 4 kByte
>> come si fa se si hanno ad es 256 k di flash?
>> organizzo la memoria in pagine da 16 k; poi ne scelgo una per volta e la carico.
>> è fondamentale conoscere la mappa della memoria perchè devo indirizzare quella.

9S12 Subsystems Interrupts

- Break in normal program execution
- Usually higher priority event
- In response to interrupt, 9S12:
  - finishes current instruction
  - stores key register values
  - selects the correct vector to be served in the Interrupt Vector Table
  - performs an Interrupt Service Routine (ISR) specific for that interrupt



-La macchina dovrà gestire anche gli interrupt esterni;
-La gestione degli interrupt non è banale e molto importante
-E' difficile programmare con gli interrupt perchè essi sono eventi asincroni.

CPU12, Interrupts Vector table

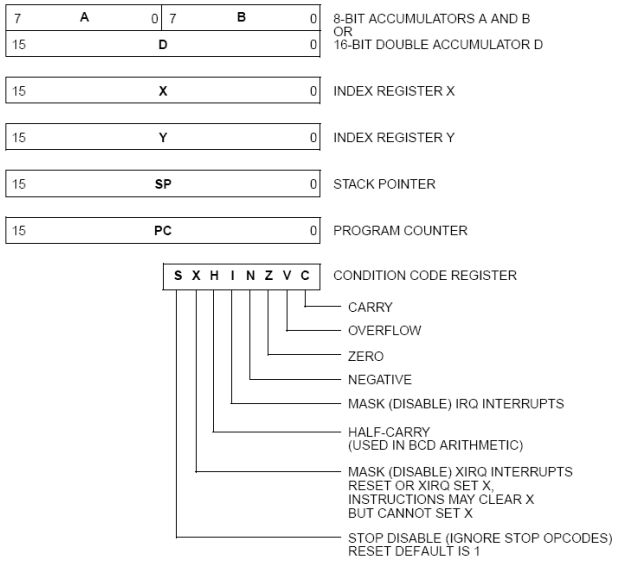
Vector Address	Interrupt Source	CCR Mask	Local Enable Register (Bit)	HPRIO Value to Elevate
$FFFE, $FFFF	Reset	none	none	–
$FFFC, $FFFD	COP Clock Monitor Fail Reset	none	COPCTL (CME, FCME)	–
$FFFA, $FFFB	COP Failure Reset	none	COP rate selected	–
$FFF8, $FFF9	Unimplemented Instruction Trap	none	none	–
$FFF6, $FFF7	SWI	none	none	–
$FFF4, $FFF5	XIRQ	X bit	none	–
$FFF2, $FFF3	IRQ	I bit	INTCR (IRQEN)	$F2
$FFF0, $FFF1	Real Time Interrupt	I bit	RTICTL (RTIE)	$F0
$FFEE, $FFEF	Timer Channel 0	I bit	TMSK1 (C0I)	$EE
$FFEC, $FFED	Timer Channel 1	I bit	TMSK1 (C1I)	$EC
$FFEA, $FFEB	Timer Channel 2	I bit	TMSK1 (C2I)	$EA
$FFE8, $FFE9	Timer Channel 3	I bit	TMSK1 (C3I)	$E8
$FFE6, $FFE7	Timer Channel 4	I bit	TMSK1 (C4I)	$E6
$FFE4, $FFE5	Timer Channel 5	I bit	TMSK1 (C5I)	$E4
$FFE2, $FFE3	Timer Channel 6	I bit	TMSK1 (C6I)	$E2
$FFE0, $FFE1	Timer Channel 7	I bit	TMSK1 (C7I)	$E0
$FFDE, $FFDF	Timer Overflow	I bit	TMSK2 (TOI)	$DE
$FFDC, $FFDD	Pulse Accumulator Overflow	I bit	PACTL (PAOVI)	$DC
$FFDA, $FFDB	Pulse Accumulator Input Edge	I bit	PACTL (PAI)	$DA
$FFD8, $FFD9	SPI Serial Transfer Complete	I bit	SP0CR1 (SPIE)	$D8
$FFD6, $FFD7	SCI 0	I bit	SC0CR2 (TIE, TCIE, RIE, ILIE)	$D6
$FFD4, $FFD5	Reserved	I bit		$D4
$FFD2, $FFD3	ATD	I bit	ATDCTL2 (ASCIE)	$D2
$FFD0, $FFD1	MSCAN Wake-Up	I bit	CRIER (WUPIE)	$D0
$FFCA-$FFCF	Reserved	I bit		$CA-$CF
$FFC8, $FFC9	MSCAN Errors	I bit	CRIER (RWRNIE, TWRNIE, RERRIE, TERRIE, BOFFIE, OVRIE)	$C8
$FFC6, $FFC7	MSCAN Receive	I bit	CRIER (RXFIE)	$C6
$FFC4, $FFC5	MSCAN Transmit	I bit	CTCR (TXEIE[2:0])	$C4
$FF80–$FFC3	Reserved	I bit		$80–$C3

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 31 di 334

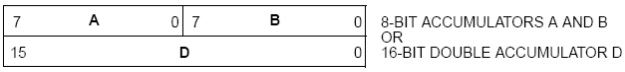
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 334

9S12 Registers (again)



I uC più semplici si basano sul concetto di accumulatore
>> tutte le operazioni matematiche e logiche passano attraverso l'accumulatore

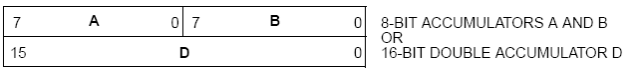
9S12 Registers - Accumulators



- Accumulators
  - Source and destination of arithmetic operations
  - A, B are 8-bit registers
  - D is the combination of A and B
    - Forms a 16-bit register
    - A is the MSB and B is the LSB
  - Used for 16-bit operations

```
ldaa #$5
```


9S12 Registers - Accumulators



Is there any difference between the following assembly code examples?

```
ldaa #$50          vs.   ldd #$0150     vs.   ldd #$5001
ldab #$01

ldaa #$00          vs.   ldd #10        vs.   clra
ldab #$0A                                     ldab #$0A
```

Freescale uses the Big Endian approach;
# means immediate ;
$ means hexadecimal ;

ldaa >> load acc a
>> immediate
$ >> hexadecimal

Mod by Giorgio Fissore

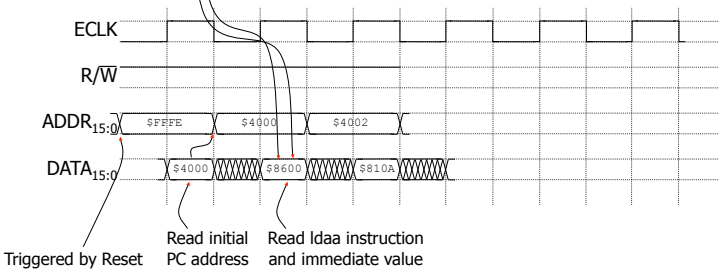
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 33 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 334

Instruction Execution Timing - Reset

$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel

```
        ldaa #0     ; Initialize j
Loop:   cmpa #10    ; Compare j to 10
        bge EndLoop ; Else !(j<10)
        adda #1     ; Increment j
        bra Loop    ; Repeat Loop
EndLoop:            ; do something else
```



Triggered by Reset — Read initial PC address — Read ldaa instruction and immediate value

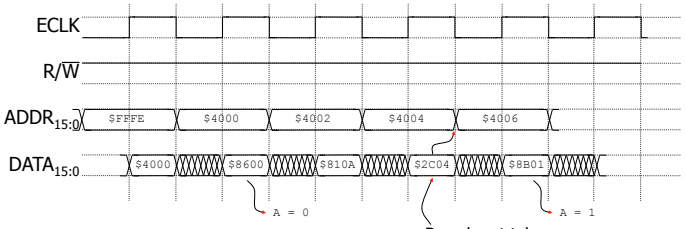
Dare un'occhiata al flusso di istruzioni

Se accendo il reset (RST), la prima istruzione sarà quella all'indirizzo FFFE ($4000) dove oltre al prox indirizzo ci sarà il codice $86 ldaa immediate adr....(???)

Instruction Execution Timing – Initial Loop Execution

$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel

```
        ldaa #0     ; Initialize j
Loop:   cmpa #10    ; Compare j to 10
        bge EndLoop ; Else !(j<10)
        adda #1     ; Increment j
        bra Loop    ; Repeat Loop
EndLoop:            ; do something else
```

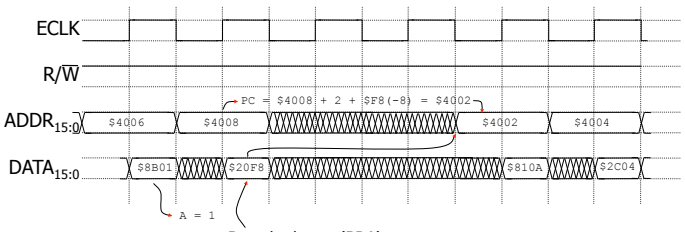


Branch not taken, requires only 1 cycles

Instruction Execution Timing – BRA Execution

$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel

```
        ldaa #0     ; Initialize j
Loop:   cmpa #10    ; Compare j to 10
        bge EndLoop ; Else !(j<10)
        adda #1     ; Increment j
        bra Loop    ; Repeat Loop
EndLoop:            ; do something else
```



Branch always (BRA) requires 3 cycles

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 35 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 334

Basic Assembly Programming

Assembly Code:

```
      ldaa #0     ; Initialize j
Loop: cmpa #10    ; Compare j to 10
      bge EndLoop ; Else !(j<10)
      staa PTT    ; Write J to PORT T
      adda #1     ; Increment j
      bra Loop    ; Repeat Loop
EndLoop:          ; do something else
```

How do we determine this value?

$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$7A	STAA extended addressing
$4007	$	Memory Location
$4008	$	
$4009	$8B	ADDA immediate addressing
$400A	$01	Value to add to A
$400B	$20	Branch Always
$400C	$F8	PC=PC+2+Rel
…		

Basic Assembly Programming

Assembly Code:

```
      ldaa #0     ; Initialize j
Loop: cmpa #10    ; Compare j to 10
      bge EndLoop ; Else !(j<10)
      staa PTT    ; Write J to PORT T
      adda #1     ; Increment j
      bra Loop    ; Repeat Loop
EndLoop:          ; do something else
```

Do the relative addresses for the BGE and BRA instructions change? Why or why not?

$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$7A	STAA extended addressing
$4007	$02	Memory Location or PORTT ($240)
$4008	$40	
$4009	$8B	ADDA immediate addressing
$400A	$01	Value to add to A
$400B	$20	Branch Always
$400C	$F8	PC=PC+2+Rel
…		

Basic Assembly Programming

Assembly Code:

```
      ldaa #0     ; Initialize j
Loop: cmpa #10    ; Compare j to 10
      bge EndLoop ; Else !(j<10)
      staa PTT    ; Write J to PORT T
      adda #1     ; Increment j
      bra Loop    ; Repeat Loop
EndLoop:          ; do something else
```

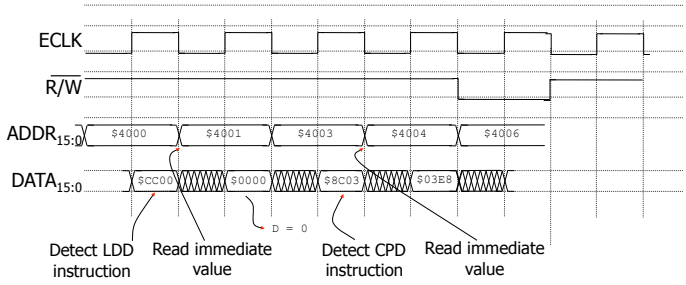
$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$07	PC=PC+2+Rel
$4006	$7A	STAA extended addressing
$4007	$02	Memory Location or PORTT ($240)
$4008	$40	
$4009	$8B	ADDA immediate addressing
$400A	$01	Value to add to A
$400B	$20	Branch Always
$400C	$F5	PC=PC+2+Rel (-11)
…		

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 37 di 334

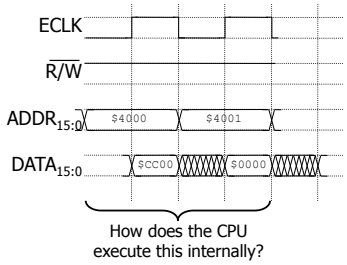
Instruction Execution Timing – Accumulator

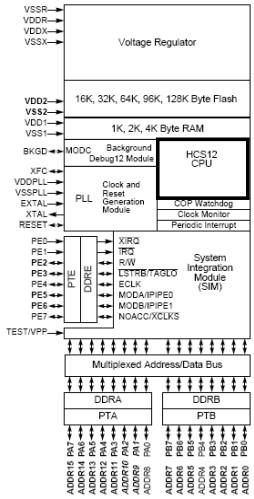
$4000	$CC	LDD immediate addressing
$4001	$00	MSB of value to be stored in D
$4002	$00	LSB of value to be stored in D
$4003	$8C	CPD immediate addressing
$4004	$03	Compare to Value 1000
$4005	$E8	



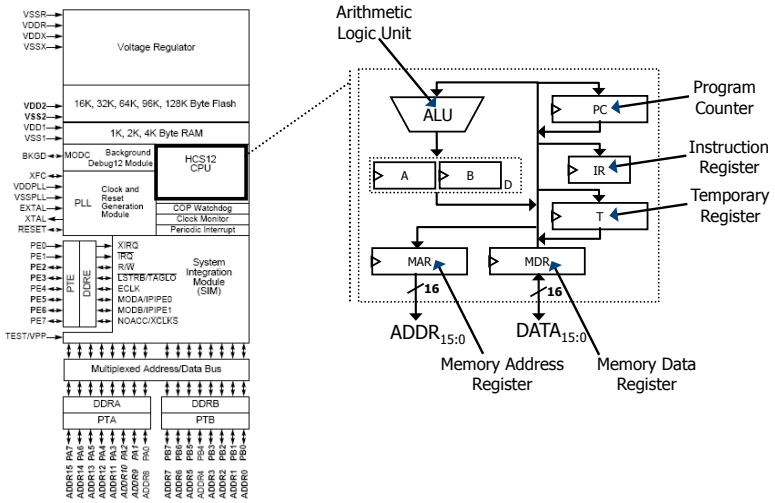
STAA Instruction Execution Timing

$4000	$CC	LDD immediate addressing
$4001	$00	MSB of value to be stored in D
$4002	$00	LSB of value to be stored in D
$4003	$8C	CPD immediate addressing
$4004	$03	Compare to Value 1000
$4005	$E8	





CPU Internals

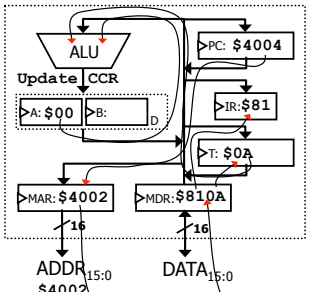


Tutto quello detto prima viene gestito dalla CPU, che è fatta solo come si vede nello schema accanto (vedi pag dopo)

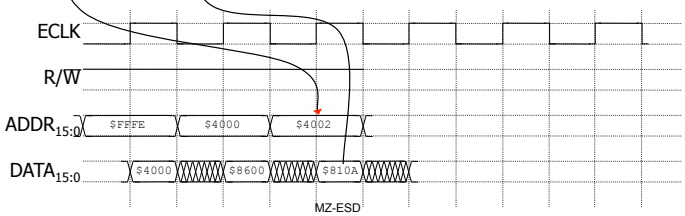
Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 39 di 334

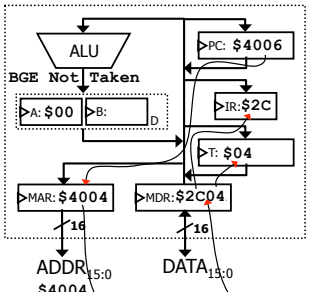
CPU Internal Execution Timing



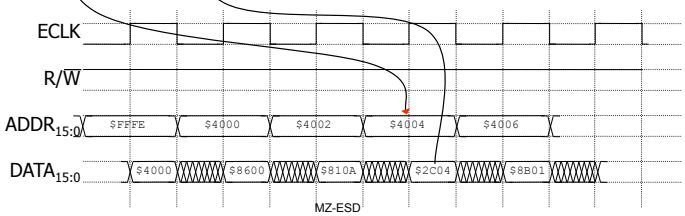
$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel



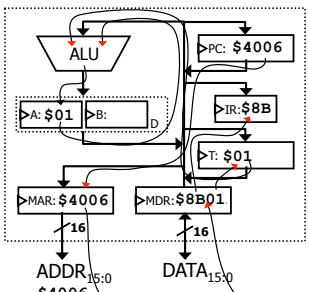
CPU Internal Execution Timing



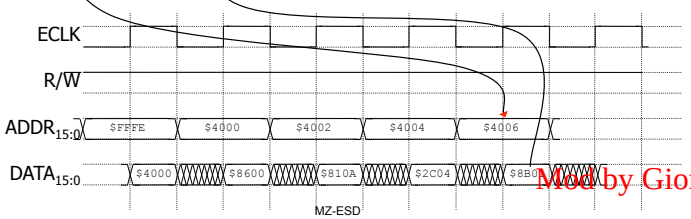
$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel



CPU Internal Execution Timing



$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel



Mod by Giorgio Fissore

HOMEWORK

ASM principles

Slide modificate da Giorgio Fissore
Disponibili in centro stampa

Valvano cap 2 pp. 27..

Se serve un riferimento sul mondo dell'assembler: capitolo 2 del Valvano

HOMEWORK

Statements and Numbers

- Source Statement is a single line of the source program.
- Source statement may contain various numbers and symbols.
- Numbers and symbols are connected by operators to make a sentence.
- Spaces, tabs are important. They serve as delimiters or separators.
- Numbers are represented by decimal, hexadecimal, or binary.

Decimal	10	A positive number
Decimal	-100	A negative number
Hexadecimal	$10	Preceded by $
Hexadecimal	$ABCD	
Binary	%1001	Binary number preceded by %

- Comment: Always use the preceding dollar or percent signs.
- The assembly process converts negative numbers to 2's complement numbers.

Assembler: unico linguaggio (pseudo-codice) parlato dalle macchine.

Le istruzioni assembler possono essere scritte solo dalla seconda colonna in poi, poichè la prima è riservata alle Label.

HOMEWORK

User Symbols

- MNEMONICS - LDAA, ldaa, STAA, staa, ...
- Symbols consists of alphanumeric characters. The 1st character must be alphabetic.
- User defined symbols may not consist of the single characters A, B, X, Y. Also do not define symbols after the mnemonics.
- Each symbol must be unique! Examples: TIME, INIT4, A444. Incorrect: 3ABC, 125, ABC
- Combination of symbols, +, -, * and / are called expressions.
- Example:
  - ABC + 1 ; add 1 to the symbol ABC

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 43 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 334

HCS12 Instruction Glossary

LDAA — Load Accumulator A — LDAA

Operation: (M) ⇒ A

Description: Loads the content of memory location M into accumulator A. The condition codes are set according to the data.

CCR Details:

S	X	H	I	N	Z	V	C
–	–	–	–	Δ	Δ	0	–

N: Set if MSB of result is set; cleared otherwise
Z: Set if result is $00; cleared otherwise
V: 0; cleared

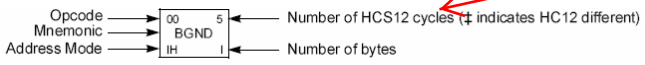
Source Form	Address Mode	Object Code	Access Detail HCS12	Access Detail M68HC12
LDAA #*opr8i*	IMM	86 ii	P	P
LDAA *opr8a*	DIR	96 dd	rPf	rfP
LDAA *opr16a*	EXT	B6 hh ll	rPO	rOP
LDAA *oprx0_xysp*	IDX	A6 xb	rPf	rfP
LDAA *oprx9,xysp*	IDX1	A6 xb ff	rPO	rPO
LDAA *oprx16,xysp*	IDX2	A6 xb ee ff	frPP	frPP
LDAA [D,*xysp*]	[D,IDX]	A6 xb	fIfrPf	fIfrfP
LDAA [*oprx16,xysp*]	[IDX2]	A6 xb ee ff	fIPrPf	fIPrfP

HOMEWORK

Viene descritta ogni istruzione, ed i flag che questa va a modificare.
>> posso controllare dei flag per capire cosa si fa

HCS12 Opcode Table

86 1 LDAA IM 2	96 3 LDAA DI 2	A6 3-6 LDAA ID 2-4	B6 3 LDAA EX 3
87 1 CLRA IH 1	97 1 TSTA IH 1	A7 1 NOP IH 1	B7 1 TFR/EXG IH 2
88 1 EORA IM 2	98 3 EORA DI 2	A8 3-6 EORA ID 2-4	B8 3 EORA EX 3
89 1 ADCA IM 2	99 3 ADCA DI 2	A9 3-6 ADCA ID 2-4	B9 3 ADCA EX 3
8A 1 ORAA IM 2	9A 3 ORAA DI 2	AA 3-6 ORAA ID 2-4	BA 3 ORAA EX 3
8B 1 ADDA IM 2	9B 3 ADDA DI 2	AB 3-6 ADDA ID 2-4	BB 3 ADDA EX 3



HOMEWORK

Ogni istruzione ha una pagina dedicata, poi sono tutte riassunte sull'opcode table.

numero di cicli macchina richiesti per realizzare l'operazione;
è il parametro che a noi interessa di più, dato che vogliamo sapere quanto tempo impiega il ns uC ad esempio ad eseguire un loop.

Basic Assembly Programming

Assembly Code:

```
      ldaa #0      ; Initialize j
Loop: cmpa #10     ; Compare j to 10
      bge EndLoop  ; Else !(j<10)
                   ; do something
      adda #1      ; Increment j
      bra Loop     ; Repeat Loop
EndLoop:           ; do something else
```

$4000	$86	LDAA immediate addressing
$4001	$00	Value to be stored in A
$4002	$81	CMPA immediate addressing
$4003	$0A	Compare Value 10
$4004	$2C	BGE (Branch if greater then or equal to zero)
$4005	$04	PC=PC+2+Rel
$4006	$8B	ADDA immediate addressing
$4007	$01	Value to add to A
$4008	$20	Branch Always
$4009	$F8	PC=PC+2+Rel
…		

HOMEWORK

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 45 di 334

Assembly Source Code Fields

HOMEWORK

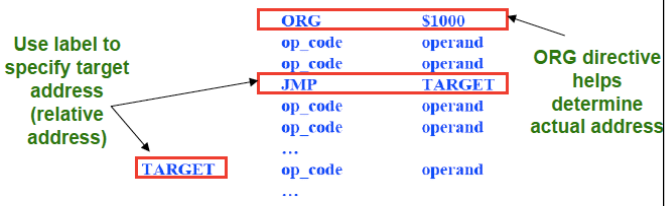
- Operation code field
  - – Contains either a mnemonic for the operation or an assembler directive or pseudo-operation
  - – Mnemonics are assembled into code to be placed into memory
  - – Assembler directives or pseudo-operations are instructions to direct the assembler how to do its job
    - – Special symbols: * and $ represent the current value of program counter
    - – Constants: Decimal (default), Hexadecimal /w ($) prefix, Binary /w (%) prefix, ASCII /w ‘ ’ and “ ”
  - – Expressions: evaluated by assembler, used for specifying constant only

Assembly Source Code Fields

HOMEWORK

- Operation code field (continued)
  - – In the following code example, the ORG (for origination) directive specifies where the code is located in memory
  - – When assembler sees the ORG directive, it sets the value of current location counter to $1000

Use label to specify target address (relative address) ORG directive helps determine actual address



Assembly Source Code Fields

HOMEWORK

- Operand field
  - – Can be: names of registers, numeric/symbolic constants, labels, algebraic expressions to be evaluated by assembler
  - – The following 2 instructions load the A, B registers from memory location DATA1 and DATA1+1

```
mov   a, DATA1
mov   b, DATA1+1
```

Label, Numeric constant

- Comment field
  - – Ignored during assembly
  - – May have source code lines that are only comments or blank
  - – For some assemblers, beginning of comment fields are denoted by “;” or “*” characters

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 47 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 334

Freescale Assembler Directives

HOMEWORK

- Discussion will now focus on the the assembler directives in the S12 Assembler Manual.
- All assembler directives are listed below, however, only the ones necessary for our present needs will be covered.

ABSENTRY - Application entry point **ALIGN - Align Location Counter**
BASE - Set number base **CLIST - List conditional assembly**
DC - Define Constant **DCB - Define Constant Block**
DS - Define Space **ELSE - Conditional assembly**
END - End assembly **ENDFOR - End of FOR block**
ENDIF - End conditional assembly **ENDM - End macro definition**
EQU - Equate symbol value **EVEN - Force word alignment**
FAIL - Generate Error message **FOR - Repeat assembly block**
IF - Conditional assembly **IFcc - Conditional assembly**
INCLUDE - Inc txt from file **LIST - Enable Listing**

1/10/2012 MZ-ESD L14/ 14

Freescale Assembler Directives

HOMEWORK

LLEN - Set Line Length **LONGEVEN - Forcing Long-Word alignment**
MACRO - Begin macro definition **MEXIT - Terminate Macro Expansion**
MLIST - List macro expansions **NOLIST - Disable Listing**
MLIST - List macro expansions **NOLIST - Disable Listing**
OFFSET - Create absolute symbols **ORG - Set Location Counter**
PAGE - Insert Page break **PLEN - Set Page Length**
RAD50 - Rad50-encoded str consts **SECTION - Declare Relocatable Section**
SET - Set Symbol Value **SPC - Insert Blank Lines**
TABS - Set Tab Length **TITLE - Provide Listing Title**
XDEF - External Symbol Definition **XREF - External Symbol Reference**
XREFB - External Reference for Symbols located on the Direct

A noi interessano le seguenti direttive:
-EQU
-ORG
-DC -define costant
-DW -define word
-DS -define storage (per riservare spazio alle variabili)
(vedi pagine dopo)

1/10/2012 MZ-ESD L14/ 15

Codewarrior Assembler Directive: Absentry

HOMEWORK

- **ABSENTRY - Application entry point**
- This directive is used to specify the application Entry Point when the Assembler directly generates an absolute file. The -FA2 assembly option - ELF/DWARF 2.0 Absolute File - must be enabled.
- Using this directive, the entry point of the assembly application is written in the header of the generated absolute file.
- When this file is loaded in the debugger, the line where the entry point label is defined is highlighted in the source window.
- This directive is ignored when the Assembler generates an object file.

Mod by Giorgio Fissore

1/10/2012 MZ-ESD L14/ X

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 49 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 334

Example: DS Directive

Show how to use the DS directive to reserve 10 bytes for data. Initialize each byte to zero in a small program segment.

```
0000              1  NUMBER:   EQU     10      ; Number of bytes allocated
0000              2  PROG:     EQU     $0800   ; Program location
0000              3  RAM:      EQU     $0900   ; Location of RAM
0800              4            ORG     PROG
                  5  ;         - - -
0800 C60A         6            ldab    #NUMBER ; Initialize B with a loop
                  7                            ; counter
0802 CE0900       8            ldx     #BUF    ; X points to the start of the
                  9                            ; buffer
0805 6930        10  loop:     clr     1,x+    ; Clear each location and
                 11                            ; point to the next location
0807 0431FB      12            dbne    b,loop  ; Decrement the loop counter
                 13                            ; and branch if the loop
                 14                            ; counter is not zero
                 15  ;         - - -
0900             16            ORG     RAM     ; Locate the data area
0900             17  BUF:      DS      NUMBER  ; Allocate the data area
```


HOMEWORK

Vediamo come lavora l'assembler per tradurre il nostro codice macchina (2 operazioni).
Riconosce il codice mnemonico delle istruzioni e lo traduce, es.
-vogliamo caricare NUMBER in acc B
>>codice operativo di ldab: es. $85
>>NUMBER: simbolo definito da qualche parte che vale 10.
=>>> L'assemblatore deve quindi:
1.generare una tabella dei simboli (l'elenco di tutti i simboli utilizzati nel programma, che però non c'entrano con il codice macchina)
>> quindi deve per prima cosa leggere tutto

Codewarrior Assembler Directive: EQU

- **EQU - Equates a symbol to a numerical value.**
  - **symbol EQU expression**
  - EQU is similar to "#define something" in C/CPP, such as,
    - #define PI 314
  - The compiler simply replaces the string PI with the number 314.
  - Example:
    - PI EQU 314
  - Makes the code:
    - LDD #PI
  - Equivalent to:
    - LDD #314
    -

HOMEWORK

Example: EQU

```
0000              1  CRLF:     EQU     $0D0A   ; For each occurrence of
                  2                            ; CRLF, the assembler will
                  3                            ; substitute the value $0D0A
0000              4  COUNT:    EQU     5       ; Loop counters often need to
                  5                            ; be initialized
0000              6  COUNT1:   EQU     COUNT*5 ; The assembler can evaluate
                  7                            ; an expression to provide
                  8                            ; a value of 25 for COUNT1
0000              9  LSMASK:   EQU     $0F     ; A mask that picks off the
                 10                            ; least significant nibble
                 11                            ; in a byte
0000             12  ls_mask:  EQU     %00001111; A binary mask equate is
                 13                            ; more readable and
                 14                            ; informative  than one
                 15                            ; given in hexadecimal
```

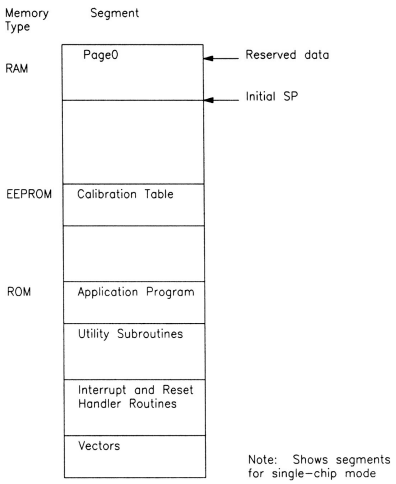
Any constant value can be defined for the assembler using the EQU

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 51 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 334

Segments



HOMEWORK

1/10/2012 MZ-ESD L14/ X

Assembly two-pass Process

- For an assembler to understand labels and symbols, it must work through the source code twice. It follows as a two-pass process.
- After the first step the assembler build a “symbol table” that gives the address of each label and symbol.

1.symbol table

2.generare il codice macchina

HOMEWORK

1/10/2012 MZ-ESD L14/ 21

Hello World! (Assembler Listing)

```
                    … … …
                     5  ; Constant equates
0000                 6  EOS:      EQU     0       ; End of string
                     7  ; Debug-12 Monitor equates
0000                 8  printf:   EQU     $FE06   ; Print a string
                     9  ; Memory map equates
0000                10  PROG:     EQU     $0800   ; RAM in the EVB
0000                11  DATA:     EQU     $0900   ; Middle of RAM
0000                12  STACK:    EQU     $0a00   ; Stack pointer
0800                13            ORG     PROG    ; Locate the program
                    14  ; Initialize stack pointer
0800 CF0A00         15            lds     #STACK
                    16  ; Print Hello World! string
0803 CC080C         17            ldd     #HELLO  ; Pass the adr of the string
0806 FEFE06         18            ldx     printf  ; The adr of the printf routine
0809 1500           19            jsr     0,x
                    20  ; Return to the monitor
080B 3F             21            swi
                    22  ; Define the string to print
080C 48656C6C       23  HELLO:    DB      'Hello World!',EOS
     6F20576F
     726C6421
     00
```

EWORK

Mod by Giorgio Fissore

1/10/2012 MZ-ESD L14/ 22

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 53 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 334

Conditional Assemblers

HOMEWORK

- There are often several conditional directives such as IFEQ MODE throughout the program, for a single directive such as MODE EQU 1.
- The single EQU directive uniformly governs all of these conditional directives. This way, a directive at the beginning of the program can control the assembly of several program segments throughout the program.
- The conditional statement argument can be a more complex expression.
- There are other conditional directives—IFNE, IFGE, IFGT, IFLE, IFLT, and IFNE—that can be used instead of IFEQ, and the ELSE statement can cause code to be assembled if the condition is false, so code immediately after the conditional is not generated.
- And conditional expressions can be nested.

9S12 Assembler Program

HOMEWORK

• **Assembler directives: Macros**

– Assembler directives MACRO and ENDM encapsulate the code

– Macro parameters: specified by \n (n is the nth parameters) in macro definition

– Labels in Marcos: A label may not occur more than once in a program. The assembler expands macro labels to make each label unique:

```
clear: MACRO
            LDX #\1
            LDAA #16
\@LOOP:  CLR 1,X+
            DBNE A,\@LOOP
ENDM
```


Example: MACRO Parameters

HOMEWORK

Example macro definition:

```
MyMacro: MACRO
DC.\0 \1, \2
ENDM
```

When this macro is used in a program, e.g.,

```
MyMacro.B $10, $56
```

the Assembler expands it to:

```
DC.B $10, $56
```

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 55 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 334

Macros and Subroutines

HOMEWORK

- Each time a macro is invoked, the assembler expands the macro “in line”.
- A subroutine code is included only once. So the macro’s make your program larger.
- A subroutine requires a call or JMP, a macro does not. So a macro is faster than a subroutine to run
- Both allows reuse of code segments. Both make the program easier to read, and allow changes to be made in one place. Both hide details of the program so you don’t need to know how it is doing something but just what it is doing

Assembler Output Files

HOMEWORK

• Assembler listing (includes a symbol table) and executable (S-Record file)
– Assembler listing
Loc [Cycles] Obj Code Line Source Line
– Symbol table: lists all symbols used in a program and their values
– Cross-reference table: shows where the symbols are defined and referenced

Final questions

HOMEWORK

- What is an assembler tool? How many passes are required? What is a Symbol Table?
- What macro are? And the conditional assembly?
- What are included in the listing file? Which are the main fields?
- Which is the format of ASM instructions?
- what about pseudo-instuctions? EQU? ORG?
- Why sections are required and how to define them?

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 57 di 334

Software Design

- Means designing the software before writing the program code
- The general approach is to learn the instruction set and the syntax first, without too much design
- As you become familiar with the processor, work on designing the solution, rather than just coding the solution
- Designing the software is more than just writing the software!

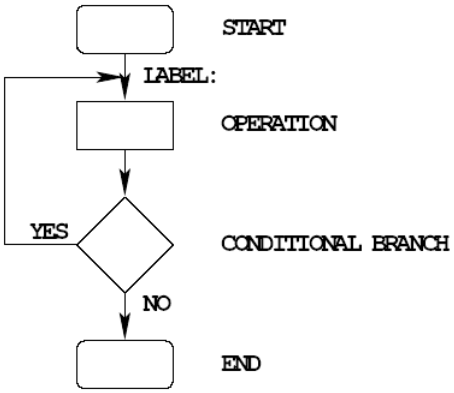
HOMEWORK

Il SW va progettato in maniera simile a come facciamo per l'HW
>> non buttarsi mai direttamente sulla scrittura del codice.
(il capitolo 5 del valvano spiega molto bene come realizzare del buon SW)

Valvano cap 5 pp. 152...

Use Flowcharts to Plan Program Structure

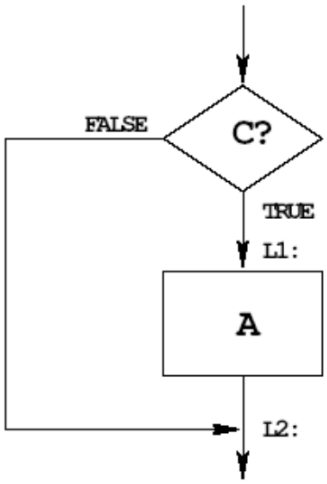
- Flow Chart Symbols:



HOMEWORK

IF-THEN Flow Structure

```
if (C)
{
  A;
}
```



HOMEWORK

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 334

Example: IF-THEN-ELSE Decision

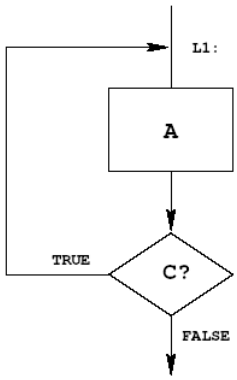
HOMEWORK

```
; Get Temperature
ldaa AD_PORT
; IF Temperature > Allowed Maximum
cmpa #MAX_ALLOWED
bls ELSE_PART ; branch lower or same (u.b.)
; THEN Turn the water valve off
ldaa VALVE_OFF
staa VALVE_PORT
bra END_IF
; ELSE Turn the water valve on
ELSE_PART:
ldaa VALVE_ON
staa VALVE_PORT
END_IF:
```


DO-WHILE Flow Structure

HOMEWORK

```
do
{
  A;
}
while (C);
```



Example: DO-WHILE

HOMEWORK

Pseudocode

```
i = 0;
do
{
   table[i] = table[i]/2;
   i = i+1;
}
while (i <= LEN);
```

Assembly Code

```
      LDX   #table
      CLRA
L1:   ASR   1,X+
      INCA
      CMPA  #LEN
      BLE   L1
```

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 61 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 334

Top-Down Design: An Example

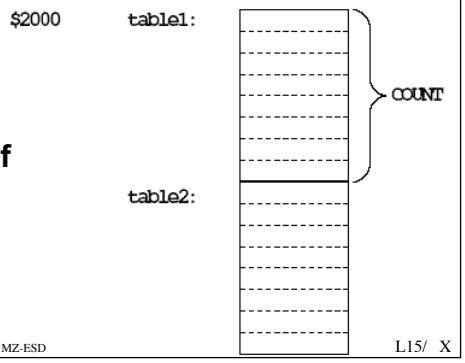
HOMEWORK

- Problem: Start with a table of data. The table consists of 5 values. Each value is between 0 and 255.
- Create a new table whose contents are the original table divided by 2

Top-Down Design: An Example

HOMEWORK

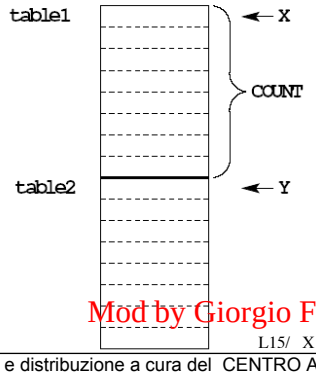
- Step 1: Determine where code and data will go in memory
  - – E.g. Code at $1000, data at $2000
- Step 2: Determine type of variables to use
  - – Because data will be between 0 and 255, can use unsigned 8-bit numbers
- Step 3: Draw a picture of the data structures in memory



Top-Down Design: An Example

HOMEWORK

- Strategy: Because we are using a table of data, we will need pointers to each table so we can keep track of which table element we are working on
  - – use the X and Y registers as pointers to the tables



Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 63 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 334

Top-Down Design: An Example

HOMEWORK

- **Step 6: Write Program**

```
; Program to divide a table by two
; and store the results in memory
prog:     equ $1000
data:     equ $2000
count:    equ 5
          org    prog      ; Set program counter to 0x2000
          ldaa   #count    ; Use A as counter
          ldx    #table1   ; Use X as data pointer to table1
          ldy    #table2   ; Use Y as data pointer to table2
l1:       ldab   0,x       ; Get entry from table1
          lsrb             ; Divide by two (unsigned)
          stab   0,y       ; Save in table2
          inx              ; Increment table1 pointer
          iny              ; Increment table2 pointer
          deca             ; Decrement counter
          bne    l1        ; Counter != 0 => more entries to divide
          swi              ; Done

          org data
table1:   dc.b $07,$c2,$3a,$68,$f3
table2:   ds.b count
```


Top-Down Design: An Example

HOMEWORK

- **Step 7: Optimize program to make use of instructions set efficiencies**

```
; Program to divide a table by two
; and store the results in memory
prog:     equ $1000
data:     equ $2000
count:    equ 5
          org    prog      ; Set program counter to 0x1000
          ldaa   #count    ; Use A as counter
          ldx    #table1   ; Use X as data pointer to table1
          ldy    #table2   ; Use Y as data pointer to table2
l1:       ldab   0, x+     ; Get entry from table1; then inc pointer
          lsrb             ; Divide by two (unsigned)
          stab   0, y+     ; Save in table2; then inc pointer
          dbne a, l1       ; Decrement counter; if not 0, more to do
          swi              ; Done
          org data
table1: dc.b $07,$c2,$3a,$68,$f3
table2: ds.b count
```


C programming and ASM comparison

HOMEWORK

Programming the HC12 in C

• A comparison of some assembly language and C constructs

Assembly	C
; Use a name instead of a num	/* Use a name instead of a num */
COUNT: EQU 5	#define COUNT 5
;----------	/*----------*/
;start a program	/* To start a program */
org $1000	main()
lds #0x3C00	{
	}
;----------	/*----------*/

• Note that in C, the starting location of the program is defined when you compile the program, not in the program itself.

• Note that C always uses the stack, so C automatically loads the stack pointer for you.

Quando si inizia un programma in assembler il "main ()" viene tradotto in una direttiva più un'istruzione:
-org $locazione: decide dove sarà fisicamente mappato il codice che stiamo eseguendo.
-lds #valore: carica nello stack un certo valore.
>>>>>> devo sempre dire dove inizia il programma e caricare il primo valore nello stack.

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 65 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 334

C programming and ASM comparison

Here is a simple program written in C and assembly. It simply divides 16 by 2. It does the division in a function.

Assembly		C
	org $2000	unsigned char i;
i:	ds.b 1	
		unsigned char div(unsigned char j);
	org $1000	main()
	lds #$3C00	{
	ldaa #16	i = div(16);
	jsr div	}
	staa i	
	swi	
div:	asra	unsigned char div(unsigned char j)
	rts	{
		return j >> 1;
		}

HOMEWORK

C programming and ASM comparison

A software delay

- **To enter a software delay, put in a nested loop, just like in assembly.**
- **Write a function delay(num) which will delay for *num milliseconds***

"voglio una funzione C che dica al uP di aspettare un determinato num di ms"

```
void delay(unsigned short num)
{
        Volatile unsigned short i;   /* variable not optimized by compiler */

        while (num > 0){

                i = xxxx;

                                     /* ------------------------------ */
                while (i > 0){       /* Want inner loop to delay       */
                        i = i - 1;   /* for 1ms                        */
                }                    /*------------------------------- */
                num = num - 1;
        }
}
```

[ovviamente quello a lato non è il modo migliore per farlo, dato che si spreca potenza, non si può fare altro nel

- **What should xxxx be to make a 1 ms delay?**

HOMEWORK

C programming and ASM comparison

- Look at assembly listing generated by compiler:

```
19: void delay(unsigned short num)
20: {
0000 6cac      [2]     STD 4,-SP
21:     volatile unsigned short i;
22:
23:     while (num > 0)
0002 2015      [3]     BRA *+23      ;abs = 0019
------------------------------------------------------
24:     {
25:             i = D_1MS;
0004 cc0736    [2]     LDD #XXXX
0007 6c82      [2]     STD 2,SP
26:             while (i > 0)
0009 2005      [3]     BRA *+7 ;abs = 0010
      ------------------------------------------------
27:             {
28:                     i = i - 1;
000b ee82      [3]     LDX 2,SP
000d 09        [1]      DEX
000e 6e82      [2]      STX 2,SP
0010 ec82      [3]      LDD 2,SP
0012 26f7      [3/1]   BNE *-7 ;abs = 000b
29:             }
      ------------------------------------------------
30:             num = num - 1;
0014 ee80      [3]      LDX 0,SP
0016 09        [1]      DEX
0017 6e80      [2]      STX 0,SP
0019 ec80      [3]      LDD 0,SP
001b 26e7      [3/1]   BNE *-23 ;abs = 0004
31:     }
32: }
------------------------------------------------------
001d 1b84      [2]     LEAS 4,SP
001f 3d        [5]     RTS
```

inner loop takes 12/10 cycles

outer loop

10 cicli quando sto dentro, 12 quando esco.

Mod by Giorgio Fissore

HOMEWORK

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 67 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 334

C programming and ASM comparison

HOMEWORK

• To clear a particular bit of a register (e.g., clear Bit 5 of PORTA) while leaving the other bits unchanged do a bitwise AND of the register and a mask which has a 0 in the bit(s) you want to clear, and a 1 in the other bits. You can construct this mask by complementing a mask which has a 1 in the bit(s) you want to set, and a 0 in the other bits:

bclr PORTA,#$20 | PORTA = PORTA & 0xDF;

or | PORTA = PORTA & ~0x20;

• To change several bits of a register, AND the register with 1's in the bits you want to leave unchanged, then OR the result with 1's in the bits you want to set, and 0's in the bits you want to clear. For example, to set bits 2 and 0, and clear bit 1 (write 101 to bits 2-0) of TSCR2, do the following:

bclr TSCR2,#$02 | TSCR2 = (TSCR2 & ~0x02) | 0x05;
bset TSCR2,#05 |

• Write to all bits of a register when you know what all bits should be, such as when you initialize it. Set or clear bits when you want to change only one or a few bits and leave the others unchanged.

Global and Local Variables

- Microcontrollers usually have ROM (FLASH) and RAM sections.
- Typically RAM sectors are small so care must be taken in variable allocation.
- Three main Variable types are allowed:
  - Global variables:
  - Static variables
  - Local variables
- Global and static variables are initialized to 0 and allocated in RAM in the same order of declaration (inside the same SECTION)
- Local variables are initialized and are allocated into the STACK; their life is limited to the function where they are declared
- If local variable is declared "static" it becomes global but its visibility is limited to the .c file (global variables can be read in all the other files linked together)

Nel C abbiamo a che fare con variabili:
-locali
-globali
-statiche
Tutte le variabili, per forza di cose finiranno in ram
Nella ram ho una sezione, definita da org, in cui metto le variabili, che saranno tipicamente scritte nell'ordine con cui le dichiaro.
Le variabili statiche sono locali, ma rimangono salvate da un utilizzo all'altro di

Tipicamente i uC hanno tanta rom e poca ram, poichè devono contenere tanto codice, ma pochi dati.

Sample C program

a. A C Program

```
0000095B 1B9D          LEAS   -3,SP

0000095D 793800        CLR    $3800

00000960 C605          LDAB   #5
00000962 87            CLRA
00000963 7C3801        STD    $3801

00000966 6A82          STAA   2,SP

00000968 C607          LDAB   #7
0000096A 6C80          STD    0,SP

0000096C 1B83          LEAS   3,SP
0000096E 3D            RTS
```

b. Assembly Language Developed From Part (a)

```
        ORG $3800  ; put global data at the beginning of RAM
guc:    DS.B 1     ; allocate a byte for scalar char variable guc
gsi:    DS.W 1     ; allocate two bytes for scalar int variable gsi
```

c. Declarations for Part (b)

Guardare questa pagina e le varie istruzioni per capire come viene tradotto il C in assembler.

```
unsigned chat guc; int gsi;
main () {
        char lsc; unsigned lui;
        guc = 0; gsi = 5;
        lsc = 0; lui = 7;
        }
```

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 69 di 334

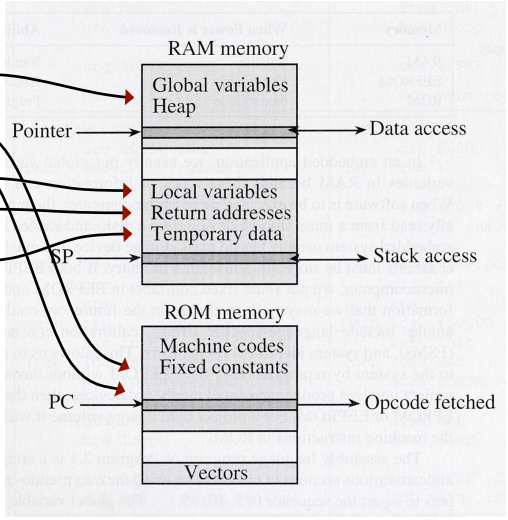
C code example produced using Code Warrior

```
#include <hidef.h> // Common defines and macros
#include <mc9s12e128.h> // Derivative information
# pragma LINK_INFO DERIVATIVE \SampleS12"
# pragma CODE_SEG _NEAR_SEG NON_BANKED
//---------------------------------------------------------------------
// Peripheral Initialization Routine
//---------------------------------------------------------------------
void PeriphInit(void)
{
DAC0_DACDRight = 0x00; // Sets DAC data to 0
DAC0_DACC0 j= 0x89; // Enables DAC:DACE =1,DSGN = 1, DACOE=1
}
//---------------------------------------------------------------------
// Main
//---------------------------------------------------------------------
void main(void)
{
        unsigned char up_dir = 1;
        unsigned long i;
        PeriphInit();        // Microcontroller initialization
        EnableInterrupts; // Enables interrupts
        for(;;)              // Forever
                {
                for(i = 0; i < 400; i++)    // Software delay ;
                        if(up_dir)
                        {
                                DAC0_DACDRight++;          // Increases the DAC output
                                if(DAC0_DACDRight == 0xFF)
                                up_dir = 0;
                        }
                        else
                        {
                                DAC0_DACDRight-;           // Decreases the DAC output
                                if(DAC0 DACDRight == 0x60)
                                up_dir = 1;
                        }
                }
}
```


Noi andiamo a scrivere e leggere porte (ce ne sono 1024).
Con gli #include all'inizio andiamo a definire da un programma già scritto i nomi di tutte le porte.

Assembly Programming – Data Organization

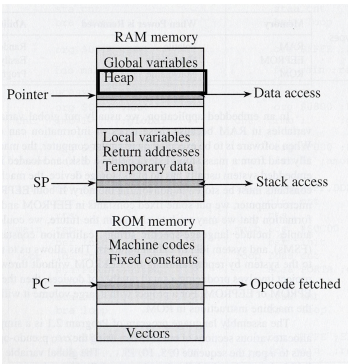
- Data Organization
  - Global Variables
  - Constants
  - Program Code
  - Local Variables
  - Return Addresses
  - Temporary Data



Dove mettiamo i dati?

Assembly Programming – Data Organization

- Data Organization
  - Heap
    - Block of memory for allocating storage whose lifetime is not related to the execution of the current routine
    - Often used for dynamically allocated data (malloc)
    - Typically located at lowest memory address after global variables



La heap è una struttura di memoria dove immagazzino dati che non sono noti durante la compilazione, ma che saranno decisi durante l'esecuzione (malloc)
>> pochissimo usati nei uC

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 71 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 334

Assembly Programming – Stack Pointer

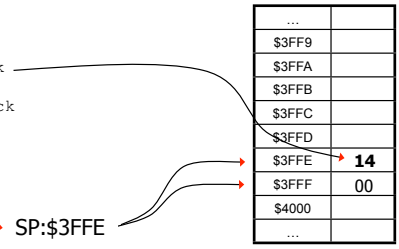
HOMEWORK

- Storing and Retrieving from top of Stack
  - Utilized stack to store temporary data
  - Push data onto stack
  - Pop data from stack
- Example:

```
ldaa #0  ; A = 0
psha     ; Push A(0) onto stack
ldaa #20 ; A = 20
psha     ; Push A(20) onto stack
ldd #300 ; D = 300
pshd     ; Push D(300) onto stack

...

puld     ; D = 300
pula     ; A = 20
pulb     ; B = 0
```



Assembly Programming – Stack Pointer

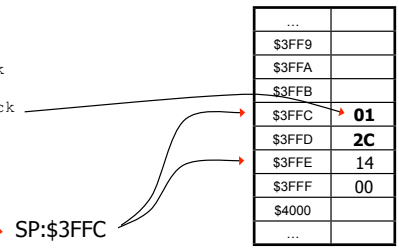
HOMEWORK

- Storing and Retrieving from top of Stack
  - Utilized stack to store temporary data
  - Push data onto stack
  - Pop data from stack
- Example:

```
ldaa #0  ; A = 0
psha     ; Push A(0) onto stack
ldaa #20 ; A = 20
psha     ; Push A(20) onto stack
ldd #300 ; D = 300
pshd     ; Push D(300) onto stack

...

puld     ; D = 300
pula     ; A = 20
pulb     ; B = 0
```



Assembly Programming – Stack Pointer

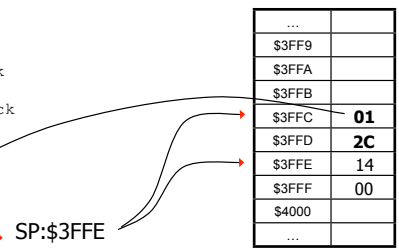
HOMEWORK

- Storing and Retrieving from top of Stack
  - Utilized stack to store temporary data
  - Push data onto stack
  - Pop data from stack
- Example:

```
ldaa #0  ; A = 0
psha     ; Push A(0) onto stack
ldaa #20 ; A = 20
psha     ; Push A(20) onto stack
ldd #300 ; D = 300
pshd     ; Push D(300) onto stack

...

puld     ; D = 300
pula     ; A = 20
pulb     ; B = 0
```



Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 73 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 75 di 334

Assembly Programming – Arrays

HOMEWORK

Array For Loop Example:

```
unsigned short a[10];
for(j=0; j<10; j++)
{
   if(a[j]==1) PORTT=0x04;
   else PORTT=0x00;
}
```

Programming Steps:

1. Initialize J
2. Compare J to 10
3. If Not Less than 10,
   1. End Loop
2. Else
   1. Load a[j]
   2. If a[j] == 1
      1. PORT T = 4
   2. Else
      1. PORT T = 0
   3. Increment J
   3. Repeat Loop (Step 2)

Assembly Code:

Initialize stack pointer

```
        lds #$4000   ; Initialize SP
        ldaa #0      ; Initialize J
        ldx #$3800   ; Initialize index to A[0]
Loop:   cmpa #10     ; Compare J to 10
        bge EndLoop  ; Else !(J<10)
        psha         ; Push J to stack
        ldd 0,X      ; load A[J]
        cpd #1       ; Compare J to 1
        bne Else     ; Else !(A[J]==1)
        ldab #4      ; Value to write to PORT T
        bra EndIf
Else:   ldab #0      ; Value to write to PORT T
EndIf:  stab PTT     ; Write value to PORT T
        pula         ; Pull(Pop) J from Stack
        adda #1      ; Increment J
        inx          ; Increment A[J]
        inx          ; Need to increment by 2
        bra Loop     ; Repeat Loop
EndLoop:             ; do something else
```

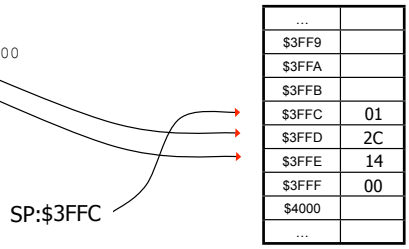
Replace ldaa and staa with psha and pula.

Assembly Programming – Stack Pointer

HOMEWORK

- Accessing items within stack (not at the top)
  - Push/Pull only provide access to top of stack
  - Can use index addressing to access stack items not at top of stack

```
...
ldab 1,SP  ; B = Mem[SP+1] = $2C
ldd  2,SP  ; D = Mem[SP+2] = $1400
...
```



...	
$3FF9	
$3FFA	
$3FFB	
$3FFC	01
$3FFD	2C
$3FFE	14
$3FFF	00
$4000	
...	

Top-Down approach to problem solving

- ✓ Algorithm
  - ➡ step by step description of how to solve a problem
- ✓ Divide and Conquer paradigm
  - ➡ the key to solve a problem is to break it down in smaller pieces (top-down approach)
- ✓ Structured Programming
  - ➡ process of breaking down a program in smaller modules

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 75 di 334

Assembly Programming – Subroutines

HOMEWORK

Instructions Related to Subroutine Calls (cont.)

[<label>]	BSR	<rel>	[<comment>]	; branch to subroutine
[<label>]	JSR	<opr>	[<comment>]	; jump to subroutine
[<label>]	RTS		[<comment>]	; return from subroutine
[<label>]	CALL	<opr>	[<comment>]	; to be used in expanded memory
	RTC			; return from CALL

where

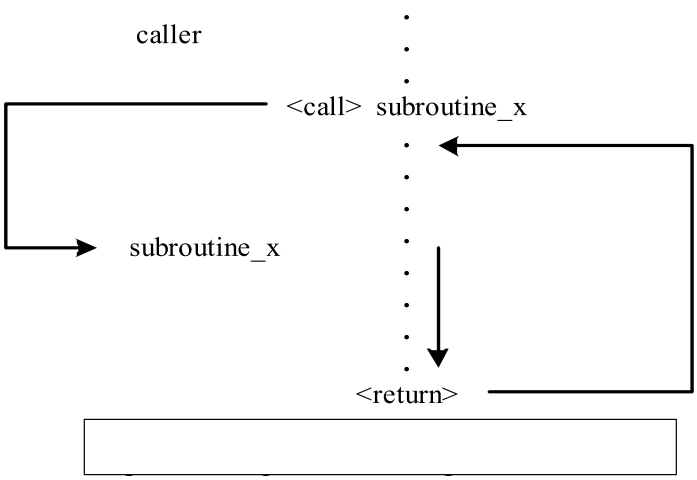
<rel> is the offset to the subroutine

<opr> is the address of the subroutine and is specified in the direct, extended, Indexed, or indexed indirect addressing modes.

Assembly Programming – Subroutines

HOMEWORK

Subroutine Processing



Subroutine operation

- ✓ Typically, a subroutine operation requires transferring and processing data
- ✓ The information-passing process occurring during the operation of a subroutine is called parameter passing
- ✓ The simplest strategy to hold a subroutine input/ output data (parameters) is using the CPU registers.
- ✓ What if there are not enough registers for holding the subroutine's input/output data?

Dove metto i dati in ingresso?
1) Nei registri della CPU (ma qui c'è poco spazio);
2) Nello stack.

Mod by Giorgio Fissore

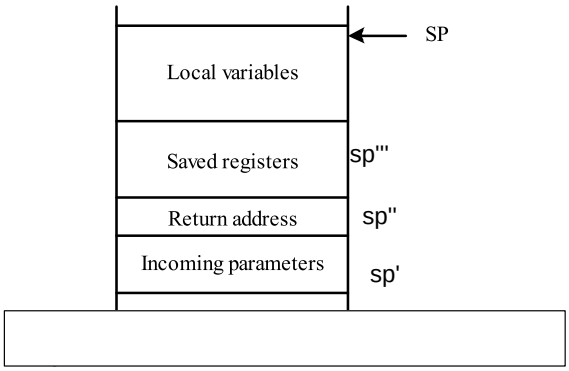
© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 77 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 334

Assembly Programming –Subroutines

Stack Frame

- The region in the stack that holds incoming parameters, the subroutine return address, local variables, and saved registers is referred to as stack frame.
- The stack frame is also called **activation record**.

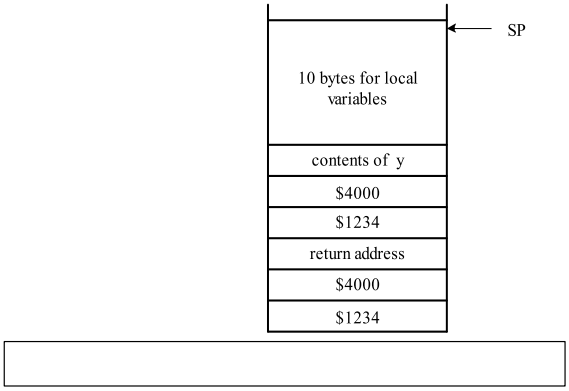


-Prima di chiamare una subroutine carico nello stack i parametri
-lo SP (stack pointer) viene spostato da SP' a SP''
-poi la prima cosa che fa la sub è salvare l'indirizzo di ritorno, e dopo sp'' > sp''', salvo cosa c'era nei registri della cpu, e vado poi a lavorare con le variabili locali

Assembly Programming –Subroutines

Example Draw the stack frame for the following program segment after the leas –10,sp instruction is executed:

```
        org $1800
        ldd     #$1234
        pshd
        ldx     #$4000
        pshx
        jsr     sub_xyz
        ldaa    $3000
        …
        swi
        org $2000
sub_xyz pshd
        pshx
        pshy
        leas    -10,sp
        …
        rts
```



Solution:
The stack frame is:

Assembly Programming –Subroutines

- Simple Subroutine Example

C Function Example:

```
for(j=0; j<10; j++)
{
   FuncA();
}

void FuncA() {
   // do something
}
```

Assembly Code:

```
      ldaa #0      ; Initialize j
Loop: cmpa #10     ; Compare j to 10
      bge EndLoop  ; Else !(j<10)
      bsr FuncA    ; Call FuncA
      adda #1      ; Increment j
      bra Loop     ; Repeat Loop
EndLoop:

FuncA:             ; do something
      rts          ; Return from FuncA
```

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 79 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 81 di 334

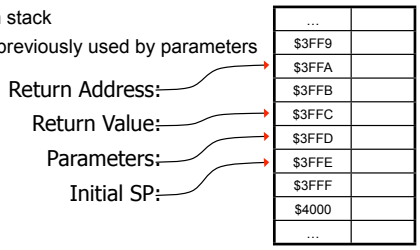
Assembly Programming –Subroutines

- Subroutines with Parameters
  - Utilize stack to pass parameters (in addition to return address)
  - Caller Initialize Steps:
    - Push function parameters onto stack
    - Allocate space on stack for return value
    - Call subroutine
  - Caller Steps:
    - Access parameters on using index addressing
  - Caller Finishing Steps:
    - Pull (Pop) return value from stack
    - Deallocate space on stack previously used by parameters

C Function Example:

```
for(j=0; j<10; )
{
   j = FuncA(j);
}

byte FuncA(byte a) {
   return a+2;
}
```



Assembly Programming –Subroutines

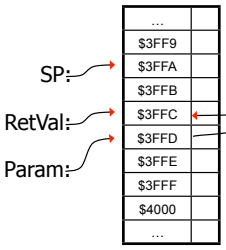
C Function Example:

```
for(j=0; j<10; )
{
   j = FuncA(j);
}

byte FuncA(byte a) {
   return a+2;
}
```

Assembly Code:

```
      ldaa #0      ; Initialize j
Loop: cmpa #10     ; Compare j to 10
      bge EndLoop  ; Else !(j<10)
      psha         ; Push parameter J onto Stack
      des          ; Allocate space for return value
      bsr FuncA    ; Call FuncA
      pula         ; Pull return value from stack
      ins          ; Deallocate space from stack
      bra Loop     ; Repeat Loop
EndLoop:
```




```
FuncA:
      ldaa 3,SP    ; Load parameter from stack
      adda #2      ; Add 2 to A
      staa 2,SP    ; Store return value to stack
      rts          ; Return from FuncA
```


Another Example using the stack

* Program that squares all the numbers in addresses $C000 to $C07F

```
        ORG $E000     ; start address
main:   LDS #$FF      ; define the stack
        LDX #$C000    ; init data block pointer
loop:   JSR square    ; squaring loop
        INX           ; point next data
        CPX #$C080    ; squared all data ?
        BNE loop      ; get more if not
here:   bra here      ; stop program
```

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 81 di 334

Parallel I/O

Slide modificate da Giorgio Fissore
Disponibili in centro stampa

Il uC è in grado di scambiare info con il mondo, e questo avviene tramite gli I/O; devo essere di volta in volta in grado di selezionare il periferico. (es serial, parallel, memoria,..)

Input/Output (I/O)

- The process of reading input signals and sending output signals is called I/O
- The I/O "direction" is defined with respect to the microprocessor
- input = read = receive
- output = write = transmit
- Generally, the memory interface is not classified as I/O since memory is considered to be an integral part of the system.

Memory or input/output mapping

- There are two ways to identify and select an external device:
  - memory mapping
  - I/O mapping
- In memory mapped I/O, each device has an address just like a memory location.
- The 9S12 uses memory mapped I/O.

-memory mapping: il periferico viene visto come uno o più indirizzi di memoria. (9s12)
-I/O mapping: spazio di indirizzamento esterno alla memoria (mondo intel).

La differenza tra questi due sta sostanzialmente nel SW:
-memoria e periferici sono raggiunti equivalentemente (si usano solo comandi di load mem, store mem)
-oppure servono comandi speciali per gli I/O.

Mod by Gio

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 83 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 85 di 334

Input/Output Register Types

➡ Control registers - hold instructions that regulate the operation of internal I/O devices

➡ Status registers - indicate the current status of internal I/O devices

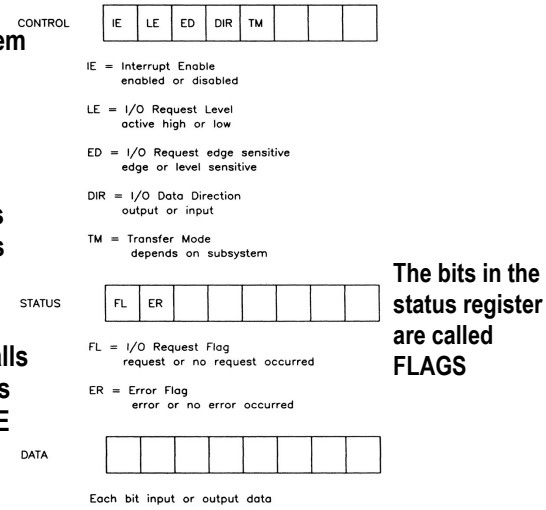
➡ Data registers - hold the input data sent to the I/O device and output data generated by this device

➡ Data direction registers - control the direction (in or out) of the data flow to/from bidirectional data registers

Questi tre tipi di reg sono tutti nella periferica e sono presenti sempre:
-la CPU da comandi alla periferica
-la periferica restituisce stati
-i dati viaggiano in entrambi i sensi, e la loro direzione viene decisa dal data direction register.

Generic I/O Subsystem Registers

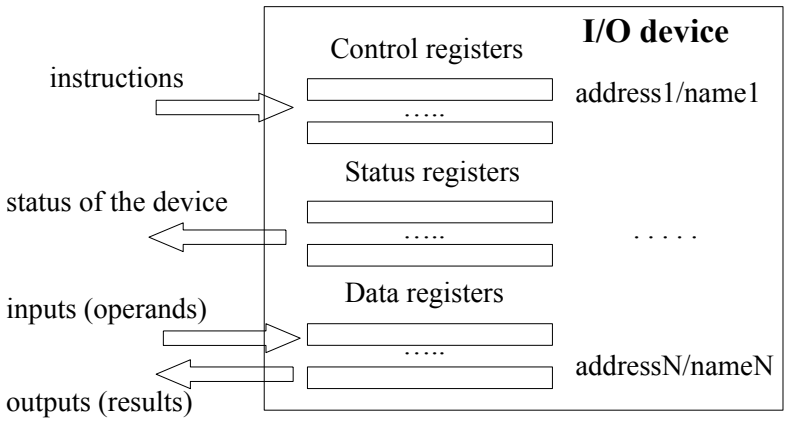
- **The parallel I/O subsystem calls the data registers PORTS**
- **The A/D subsystem calls them RESULTS registers**
- **The Timer subsystem calls them CAPTURE registers for inputs and COMPARE registers for outputs**



The bits in the status register are called FLAGS

Le porte non sono altro che registri in cui scrivo o leggo dei dati.

I/O Device Architecture



é importante che ciascuno di questi registri abbia un nome mnemonico comprensibile

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 85 di 334

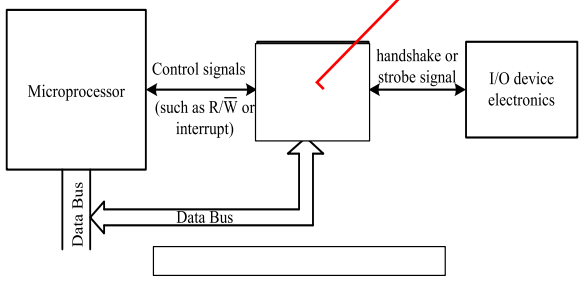
I/O Schemes

- Isolated I/O scheme
  - The microprocessor has dedicated instructions for I/O operations.
  - The microprocessor has a separate address space for I/O devices.
- Memory-mapped I/O scheme
  - The microprocessor uses the same instruction set to perform memory accesses and I/O operations.
  - The I/O devices and memory components are resident in the same memory space.

Distinguiamo ora i due casi
-Memory Mapped(il uP non si accorge di parlare con IO
-Isolated I/O (istruzioni dedicate per accedere all'IO al posto che memoria)

I/O Transfer Synchronization

- The role of an interface chip
  - Synchronizing data transfer between the CPU and the interface chip
  - Synchronizing data transfer between the interface chip and the I/O device



Questo periferico scambia le info CPU----IO
Ciò può essere fatto con vari metodi:
Polling, Interrupt

Synchronizing the Microprocessor and the Interface Chip

- The polling method
  - For input -- The microprocessor checks a status bit of the interface chip to find out if the interface chip has received new data from the input device.
  - For output -- The microprocessor checks a status bit of the interface chip to find out if it can send new data to the interface chip.
- The interrupt-driven method
  - For input -- The interface chip interrupts the microprocessor whenever it has received new data from the input device.
  - For output -- The interface chip interrupts the microprocessor whenever it can accept new data from the microprocessor.

-Polling, realizzato con un while, il uP continua a controllare un registro di stato.
-Interrupt, più intelligente e un po' più complesso, lascia il uP libero di lavorare mentre aspetta

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 334

Parallel I/O Control Register

68HC11 system:

PIOC

	B7						B0
$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB
RESET =	0	0	0	0	0	U	1	1

STAF = Strobe A (STRA) Flag
0 = Inactive
1 = Set at active edge of STRA pin

STAI = Strobe A Interrupt Enable
0 = No hardware interrupt generated
1 = Hardware interrupt requested when STAF = 1

CWOM = Port C Wire-OR Mode
0 = Port C outputs normal
1 = Open-drain

HNDS = Handshake/Simple Strobe Mode Select
0 = Simple Strobe Mode
1 = Full Handshake Mode

OIN = Output/Input Handshake Select
0 = Input
1 = Output

PLS = Pulse Mode Select for STRB Output
0 = STRB level active
1 = STRB pulses

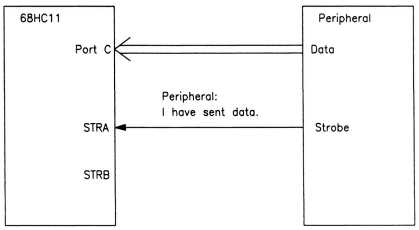
EGA = Active Edge Select for STRA
0 = High to Low (falling)
1 = Low to High (rising)

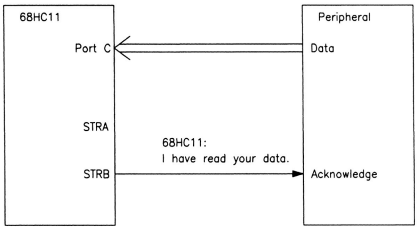
INVB = Invert STRB Output
0 = STRB active low
1 = STRB active high

1/10/2012 L16/ X

Input Handshaking Synchronization

68HC11 system:



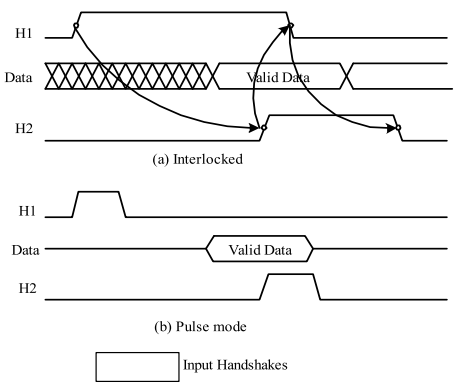


1/10/2012 MZ-ESD L16/ 19

Input Handshake Protocol

A quattro fronti

- Step 1. The interface chip asserts (or pulses) H1 to indicate its intention to input data.
- Step 2. The input device puts data on the data port pins and also asserts (or pulses) the handshake signal H2.
- Step 3. The interface chip latches the data and de-asserts H1. After some delay, the input device also de-asserts H2.



Protocollo più usato ("interleave" o "interlock").
Ne esiste ank un altro peggiore in cui si dice solo "voglio un dato, mandamelo" ma non si da più l'ACK del sì, li ho ricevuti.

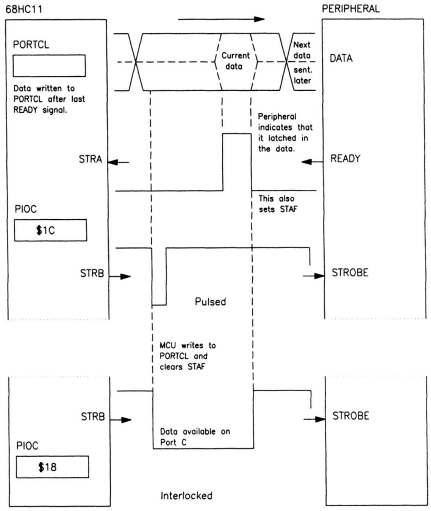
Mod by Giorgio Fissore

1/10/2012 MZ-ESD L16/ 20

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 89 di 334

Output Handshaking Timing Diagrams

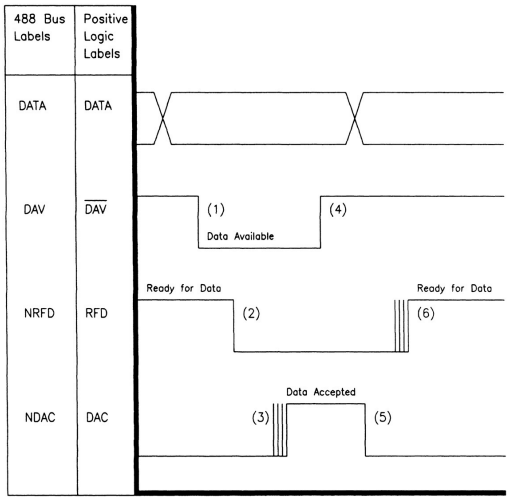
68HC11 system:



1/10/2012 MZ-ESD L16/ 24

E' un bus per interfacciare sistemi di misura

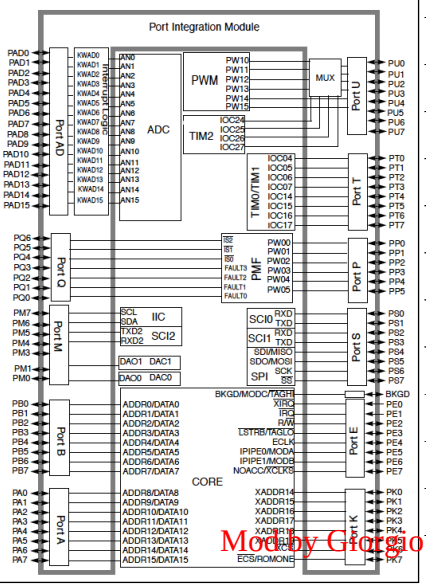
IEEE-488 Handshaking Timing Diagrams



1/10/2012 MZ-ESD L16/ 25

Overview of 9S12 Parallel Ports

- Parallel IO Ports
  - E.g. Port T, Port AD
  - Used to interface with many devices
    - Switches
    - LEDs
    - LCD
    - Keypads
    - Relays
    - Stepper Motors
  - Interface with parallel IO requires us to connect the devices correctly and write code to interface with the devices



In generale, le porte più usate sono quelle parallele;
Spesso il parallelismo è legato al byte (8 bit), anche se naturalmente non è una regola. (talvolta le porte seriali hanno un parallelismo di 2, uno IN e l'altro OUT).
Se io devo mandare dei dati su una porta la CPU deve:
1) scrivere degli opportuni comandi sui registri di controllo (istruisci il periferico su cosa deve fare)
2) scrivere i dati (dare i dati che il periferico deve girare all'esterno)

Modify by Giorgio Fissore

1/10/2012 MZ-ESI

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 93 di 334

Overview of 9S12 Parallel Ports

- The name of port data register can be formed by adding letters “PT” o “PORT” as the prefix to the port name. For example, PTA, PTB, PTP, and PTT (PORTA, PORTB...).
- Output a value to a port is done by storing that value to the port data register.

```
movb    #$FF,DDRH       ; configure Port H for output
movb    #$37,PTH        ; output the hex value 37 to port H
```

- Input a value from an input port is done by loading from the port data register.

```
movb    #0,DDRH ; configure Port H for input
ldaa    PTH     ; read data from port H into A
```

- An I/O port may have up to eight associated registers.

Ogni porta avrà un certo numero di reg di configurazione, non imparare a memoria i loro significati!!

Port A and Port B

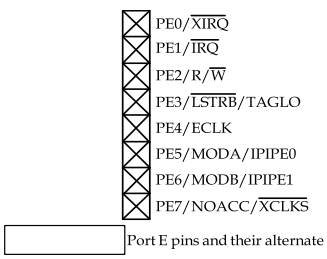
HOMEWORK

- In expanded mode, Port A carries the time-multiplexed higher address/data signals A15/D15…A8/D8.
- In expanded mode, Port B carries the time-multiplexed lower address/data signals A7/D7…A0/D0.
- In single chip mode, these two ports are used as general I/O ports.

Port E

HOMEWORK

- Port E pins are used for bus control and interrupt service request signals.
- When a Port E pin is not used as control or interrupt signal, it can be used as general I/O pin.



Port E pins and their alternate functions

La porta E può avere diverse funzioni, esempio può essere usata per gestire interrupt, se però la mia macchina non avesse int, potrebbe essere usata come IO, il tutto sempre configurando determinati registri.

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 93 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 334

Pull-Up Control Register

HOMEWORK

	7	6	5	4	3	2	1	0
	PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE
reset:	1	0	0	1	0	0	0	0

PUPKE: pull-up Port K enable
0 = Port K pull-up resistors are disabled
1 = Port K pull-up resistors are enabled
PUPEE: pull-up Port E enable
0 = Port E input pins 7 and 4-0, pull-up resistors are disabled
1 = Port E input pins 7 and 4-0, pull-up resistors are enabled
PUPBE: pull-up Port B enable
0 = Port B pull-up resistors are disabled
1 = Port B pull-up resistors are enabled
PUPAE: pull-up Port A enable
0 = Port A pull-up resistors are disabled
1 = Port A pull-up resistors are enabled

Pull-Up control register

DRRIV Register

HOMEWORK

	7	6	5	4	3	2	1	0
	RDPK	0	0	RDPE	0	0	RDPB	RDPA
reset:	1	0	0	1	0	0	0	0

RDPK: reduced drive of Port K
0 = All Port K pins have full drive enabled
1 = All Port K pins have reduced drive enabled
RDPE: reduced drive of Port E
0 = All Port E pins have full drive enabled
1 = All Port E pins have reduced drive enabled
RDPB: reduced drive of Port B
0 = All Port B pins have full drive enabled
1 = All Port B pins have reduced drive enabled
RDPA: reduced drive of Port A
0 = All Port A pins have full drive enabled
1 = All Port A pins have reduced drive enabled

Reduced Drive Register (RDRIV)

Port K

HOMEWORK

- Has PTK and DDRK registers
- Carries expanded address XADDR14…XADDR19 in expanded mode
- On the rising edge of the RESET signal, the value of the PK7 pin is latched into the ROMON bit of the MISC register in expanded mode. If it is 1, the on-chip flash memory is enabled.
  - ROMON is forced to 1 in single chip mode.

PK0/X14
PK1/X15
PK2/X16
PK3/X17
PK4/X18
PK5/X19
PK6/$\overline{XCS}$ (only available in H sub-family)
PK7/$\overline{ECS}$/ROMONE

Port K pins and their alternate functions

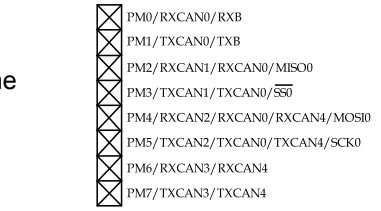
Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 95 di 334

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 334

Port M

- Port M has all the equivalent registers that Port S has and also a module routing register (MODRR).
  - The MODRR configures the rerouting of CAN0, CAN4, SPI0, SPI1, and SPI2 on defined port pins.



Port M pins and their alternate functions

HOMEWORK

MODRR Register

	7	6	5	4	3	2	1	0
	0	MODRR6	MODRR5	MODRR4	MODRR3	MODRR2	MODRR1	MODRR0
reset:	0	0	0	0	0	0	0	0

CAN0 routing

MODRR1	MODRR0	RXCAN0	TXCAN0
0	0	PM0	PM1
0	1	PM2[1]	PM3[1]
1	0	PM4[2]	PM5[2]
1	1	reserved	

CAN4 routing

MODRR3	MODRR2	RXCAN4	TXCAN4
0	0	PJ6	PJ7
0	1	PM4[3]	PM5[3]
1	0	PM6[4]	PM7[4]
1	1	reserved	

SPI0 routing

MODRR4	MISO0	MOSI0	SCK0	SS0
0	PS4	PS5	PS6	PS7
1	PM2[5]	PM4[6]	PM5[6]	PM3[5]

SPI1 routing

MODRR5	MISO1	MOSI1	SCK1	SS1
0	PP0	PP1	PP2	PP3
1	PH0	PH1	PH2	PH3

SPI2 routing

MODRR6	MISO2	MOSI2	SCK2	SS2
0	PP4	PP5	PP7	PP6
1	PH4	PH5	PH6	PH7

Note: 1. Routing to this pin takes effect only if CAN1 is disabled.
2. Routing to this pin takes effect only if CAN2 is disabled.
3. Routing to this pin takes effect only if CAN2 disabled and CAN0 disabled if routed here.
4. Routing to this pin takes effect only if CAN3 is disabled.
5. Routing to this pin takes effect only if CAN1 disabled and CAN0 disabled if routed here.
6. Routing to this pin takes effect only if CAN2 is disabled and CAN0 disabled if routed here and CAN4 disabled if routed here

Module Routing register (MODRR)

HOMEWORK

Configuration Examples

• **Example 1** Give an instruction to configure the MODRR register to achieve the following port routing:

1. CAN0: use pins PM1 and PM0
2. CAN1: use pins PM3 and PM2
3. CAN2: use pins PM5 and PM4
4. CAN3: use pins PM7 and PM6
5. I2C: use PJ7 and PJ6
6. SPI0: use pins PS7~PS4
7. SPI1: use pins PH3~PH0
8. SPI2: use pins PH7~PH4

• **Solution:** This routing requirement can be achieved by preventing CAN4 from using any port pins and keep the default routing after reset.

-The following instruction will satisfy the requirement:

```
movb   #$60,MODRR        ; CAN4 must be disabled
```

HOMEWORK

Mod by Giorgio Fissore

© Proprietà riservata dell'autore - Digitalizzazione e distribuzione a cura del CENTRO APPUNTI - Corso Luigi Einaudi, 55 - Torino / Pagina 97 di 334