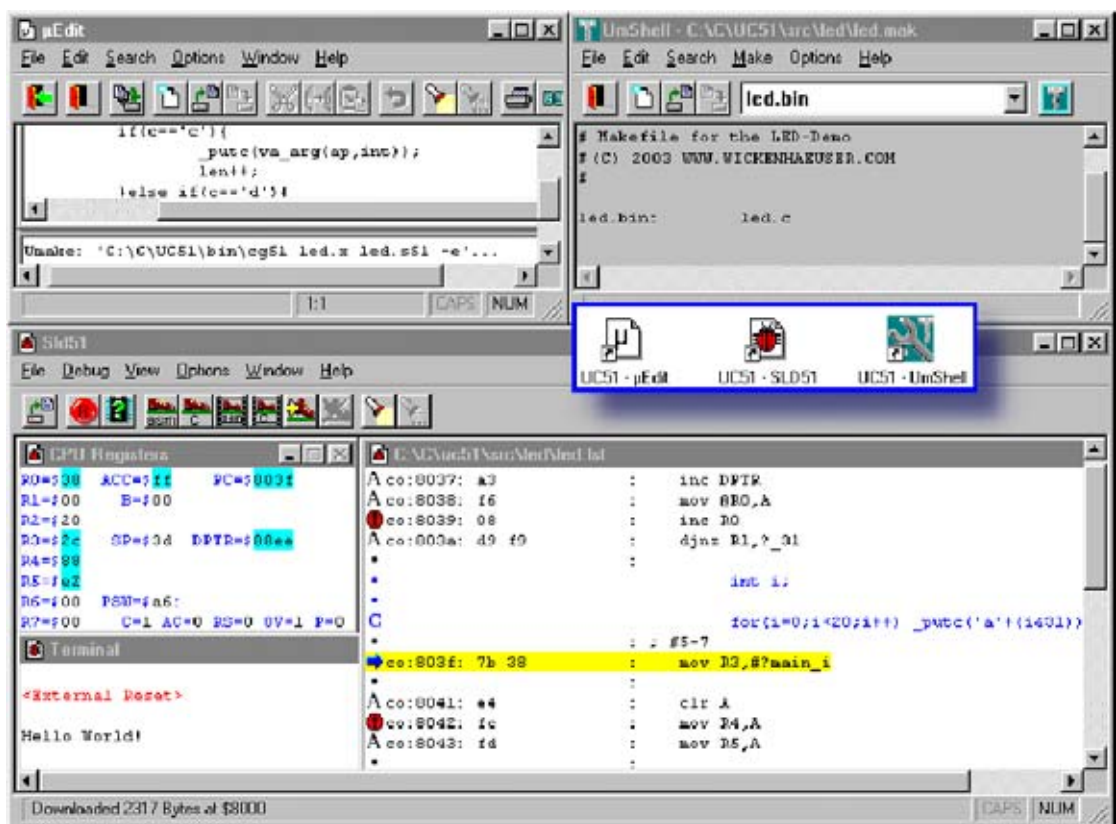


μC/51

Compilatore C per micro famiglia 51

MANUALE UTENTE



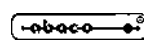
grifo[®]
ITALIAN TECHNOLOGY

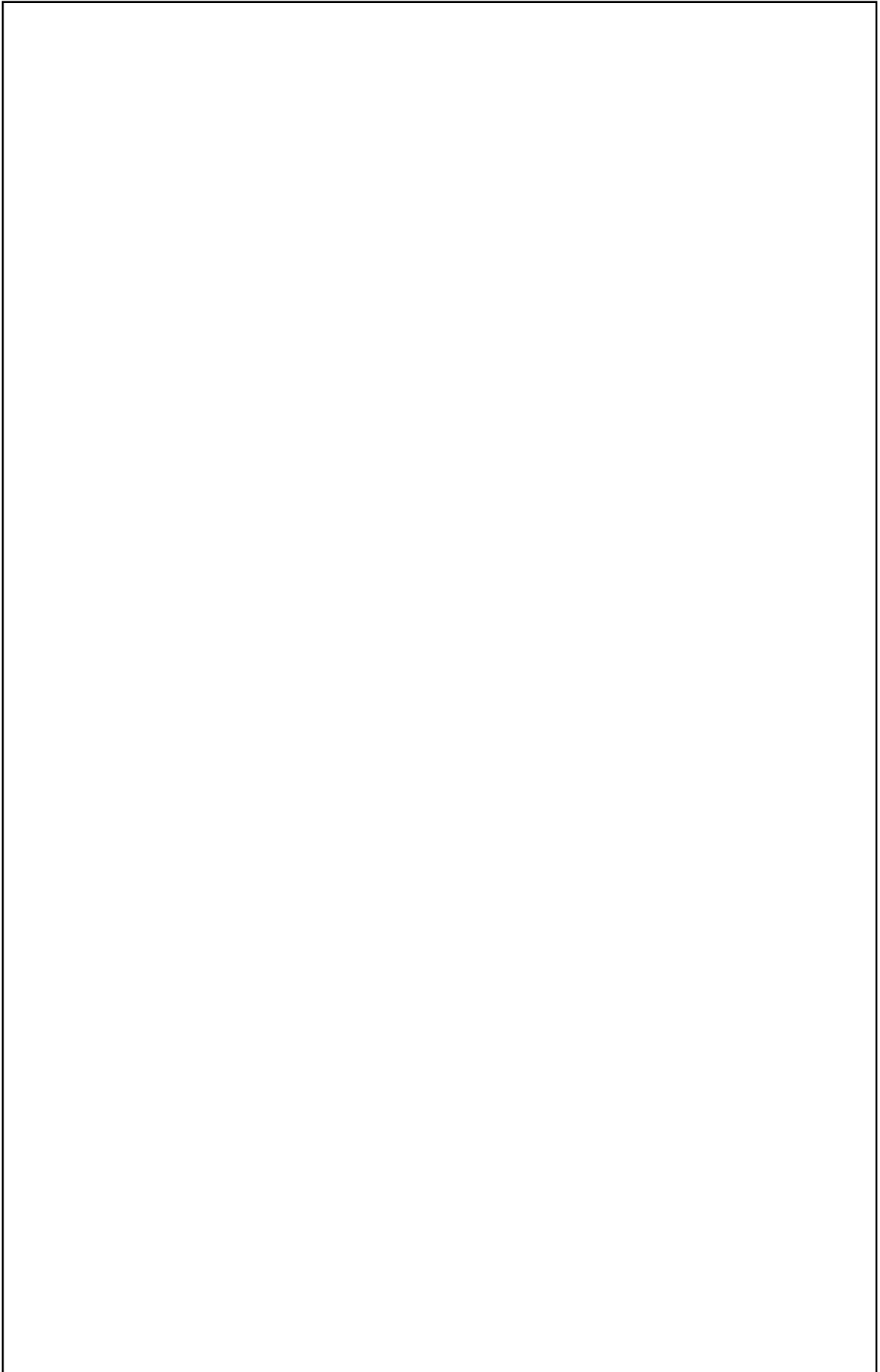
Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it



<http://www.grifo.it> <http://www.grifo.com>
Tel. +39 051 892.052 (a. r.) FAX: +39 051 893.661

μC/51 Edition 3.00 Rel. 25 February 2003

, GPC[®], grifo[®], are trade marks of grifo[®]



μC/51

Compilatore C per micro famiglia 51

μC/51 è un **compilatore** conforme alle norme **ANSI** pensato per sviluppare progetti software per i microcontrollori della **famglia 51** da PC con Windows.

La distribuzione comprende un ambiente completo di:

- Editor dei sorgenti
- Editor dei MakeFile
- Compilatore μC/51
- Assembler A51
- Linker
- Generatore di librerie
- Downloader
- **Debugger a livello di sorgente**

La **struttura modulare** permette di sostituire alcuni elementi, ad esempio l'editor dei sorgenti, con un oggetto dalla funzione analoga di propria scelta.

La gestione dei progetti è basato sul potente standard internazionale noto come **Make**, una diffusa utility storica per lo sviluppo del software in ambienti UNIX.

Il compilatore è ottimizzante ed ha le seguenti caratteristiche:

- Grafi di chiamata per minimizzare l'utilizzo della RAM.
- Ottimizzazione flusso dati.
- Istruzioni specifiche per le periferiche.
- Propagazione delle copie.
- Loop reversion.
- Loop rotation.
- Loop induction.
- Strength reduction.
- Eliminazione sottoespressioni comuni.
- Coercizione intelligente degli interi.
- Ottimizzazione "Peephole".

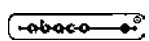
grifo[®]
ITALIAN TECHNOLOGY

Via dell' Artigiano, 8/6
40016 San Giorgio di Piano
(Bologna) ITALY
E-mail: grifo@grifo.it



<http://www.grifo.it> <http://www.grifo.com>
Tel. +39 051 892.052 (a. r.) FAX: +39 051 893.661

μC/51 Edition 3.00 Rel. 25 February 2003

, GPC[®], grifo[®], are trade marks of grifo[®]

DOCUMENTATION COPYRIGHT BY grifo®, ALL RIGHTS RESERVED

No part of this document may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, either electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written consent of **grifo®**.

IMPORTANT

Although all the information contained herein have been carefully verified, **grifo®** assumes no responsibility for errors that might appear in this document, or for damage to things or persons resulting from technical errors, omission and improper use of this manual and of the related software and hardware.

grifo® reserves the right to change the contents and form of this document, as well as the features and specification of its products at any time, without prior notice, to obtain always the best product.

For specific informations on the components mounted on the card, please refer to the Data Book of the builder or second sources.

SYMBOLS DESCRIPTION

In the manual could appear the following symbols:

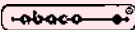


Attention: Generic danger



Attention: High voltage

Trade Marks

, **GPC®**, **grifo®** : are trade marks of **grifo®**.

Other Product and Company names listed, are trade marks of their respective companies.

INDICE GENERALE

INTRODUZIONE	1
PREFAZIONE	2
INTRODUZIONE	2
LA STORIA	2
PROCESSORI SUPPORTATI	2
QUALITA' DEL CODICE	3
LIMITI DELLA VERSIONE DEMO	4
COMPATIBILITA' DEL SORGENTE CON ALTRI COMPILATORI	4
VERSIONE COMPLETA	4
SUPPORTO	4
INFORMAZIONI SULLA GRIFO®	4
PER IL FUTURO	5
TERMINI DI UTILIZZO	5
ERRORI	5
INTERNET	5
INSTALLAZIONE E DISINSTALLAZIONE	5
AVVIO	6
INSTALLAZIONE DELLA VERSIONE COMPLETA	6
DOCUMENTAZIONE	6
AVVIO RAPIDO	8
INSTALLAZIONE	8
HARDWARE DI SVILUPPO	8
ALTERNATIVA	8
HELLO WORLD !	9
ANCORA SULLA COMPATIBILITA' ANSI	10
UN ESAME RAVVICINATO	11
DATI FONDAMENTALI DEL μ C/51	11
BUG REPORT	11
TIPI DI DATI	12
PRECISIONE IN VIRGOLA MOBILE	12
TIPI DI PUNTATORI	12
MODELLI DI MEMORIA	13
MODIFICATORI SPECIFICI PER 8051	13
SUI MODIFICATORI, AREE DI MEMORIA E SEGMENTAZIONE	14
PER RIASSUMERE	15
MODIFICATORI NELLA TYPEDEF	16
UTILIZZO DEI REGISTRI	16
INTERRUPTS IN C	16
LAVORARE CON I GRAFI DI CHIAMATA	17
UTILIZZO DEL FORMATTATORE DELLA PRINTF()	17
SULLE STRINGHE IN GENERALE	18
MISCHIARE C ED ASSEMBLER	19

UTILIZZO DELL'ASSEMBLER	19
ALCUNI SUGGERIMENTI PER USARE L'ASSEMBLER	20
FUNZIONI INDIRETTE	20
FUNZIONI CON NUMERO DI ARGOMENTI VARIABILE	21
PROMOZIONE DEGLI INTERI	21
FUNZIONI OBSOLETE	21
DEFINIRE UNO SFR, INDIRIZZI ASSOLUTI MEDIANTE '@'	21
RIDEFINIRE LE FUNZIONI DI LIBRERIA	22
LA FUNZIONE STARTUP()	22
CONTROLLO DI INTEGRITA': LA FUNZIONE '_BIN_SAFE()'	22
EFFICIENZA DEL CODICE	23
SIMBOLI PREDEFINITI	23
OPZIONI IMPORTANTI (LINEA DI COMANDO, #PRAGMA)	23
UMSHELL E UMAKE	24
I FLAGS PIU' IMPORTANTI NEI MAKEFILE	24
DESCRIZIONE DEI MAKEFILES	25
REGOLE IMPLICITE E REGOLE ESPLICITE	26
CREARE LE PROPRIE REGOLE	27
DESCRIZIONE TECNICA DELL'ASSEMBLER A51.EXE	28
MNEMONICI	28
NOMI, VARIABILI, ETICHETTE	28
NUMERI	28
OPERATORI	29
DIRETTIVE	29
MACRO	35
SIMBOLI GENERATI	35
LIBRERIE STANDARD	36
STDIO.H	36
STRING.H	37
CTYPE.H	37
STDARG.H	38
BIN_SAFE.H	38
MATH.H	38
HEADER FILES SPECIFICI DELLA FAMIGLIA 51	39
IRQ52.H	39
REG51.H, REG52.H, REG535.H	39
KAR.H	39
PORTARE DA ALTRI COMPILATORI	40
UTILIZZO DELLA MEMORIA - MODELLI DI MEMORIA	40
HEADER FILE UTILI	40
INDIRIZZI ASSOLUTI	40
INTERRUPTS	41
LINGUAGGIO ASSEMBLY	41
APPENDICE A: INDICE ANALITICO	A-1

INDICE DELLE FIGURE

FIGURA 1: I MODULI DI EDITOR E DEBUGGER A LIVELLO DI SORGENTE	7
FIGURA 2: STRUTTURA DELLA MEMORIA DELL'OS535	9
FIGURA 3: IL DOWNLOADER	27
FIGURA 4: IL DEBUGGER A LIVELLO DI SORGENTE	39





INTRODUZIONE

Scopo di questo manuale é la trasmissione delle informazioni necessarie all'uso competente e sicuro dei prodotti. Esse sono il frutto di un'elaborazione continua e sistematica di dati e prove tecniche registrate e validate dal Costruttore, in attuazione alle procedure interne di sicurezza e qualità dell'informazione.

Per un corretto rapporto coi prodotti, é necessario garantire leggibilità e conservazione del manuale, anche per futuri riferimenti. In caso di deterioramento o più semplicemente per ragioni di approfondimento tecnico ed operativo, consultare i nostri siti www.grifo.it o www.grifo.com o direttamente l'Assistenza Tecnica autorizzata.

Al fine di non incontrare problemi nell'uso di tali dispositivi, é conveniente che l'utente - **PRIMA DI COMINCIARE AD OPERARE** - legga con attenzione tutte le informazioni contenute in questo manuale. In una seconda fase, per rintracciare più facilmente le informazioni necessarie, si può fare riferimento all'indice generale e all'indice analitico, posti rispettivamente all'inizio ed alla fine del manuale.

Il programma qui descritto è coperto da diritto d'autore, tutti i diritti sono riservati. Nè il programma nè alcuna sua parte possono essere analizzati, disassemblati o modificati in alcun modo, con qualunque mezzo, per qualunque scopo.

Questo documento è coperto da diritto d'autore, tutti i diritti sono riservati. Questo documento non può essere copiato, riprodotto o tradotto in alcun modo o mediante alcun mezzo, nè per intero nè in parte, senza il permesso scritto della **grifo®**.

grifo® non si assume alcuna responsabilità per l'uso errato di questo manuale.

grifo® si riserva il diritto di apportare cambiamenti o miglioramenti ai prodotti descritti in questo manuale in qualunque momento senza darne notizia.

Questo manuale contiene nomi di aziende, software, prodotti, ecc. che sono marchi registrati dai rispettivi proprietari. **grifo®** rispetta tale diritto.

PREFAZIONE

INTRODUZIONE

Grazie per avere deciso di usare $\mu\text{C}/51$, un sistema completo per lo sviluppo di applicativi in linguaggio ANSI C dedicato all'intera famiglia di microcontrollori 8051.

Questa è la documentazione per la prima “versione ufficiale”. Sebbene l'attuale versione sia la 1.xx non si corre il rischio di “muoversi per una selva oscura”: questo sistema di sviluppo nasce da una lunga esperienza, il codice generato è molto stabile ed altamente ottimizzato.

Molti applicativi industriali sono già stati generati con questo sistema.

Le librerie, i codici sorgente e gli esempi inclusi nella presente distribuzione coprono una grande varietà di dispositivi: dal più piccolo micro 8051 disponibile (**89c1051** della ATMEL con appena 1kB di codice e 64 bytes di RAM interna) fino al più grande ($\mu\text{C}/51$ è in grado di gestire dimensioni di memoria fino a 16 MB), dal **1-Wire**® della Maxim, passando per **I²C Bus**, fino a **Ethernet**.

LA STORIA

Già parecchi anni fa gli sviluppatori sentivano la necessità di strumenti per scrivere applicativi liberandosi dalle scomodità dell'assembly o dalla lentezza dei basic interpretati. Nacquero così dei compilatori basic (ad esempio, il BASCOM 8051).

Sebbene all'inizio fossero strumenti semplici, già rendevano molto più rapido e facile lo sviluppo dei programmi, tuttavia la richiesta di complessità aumentava...

La versione beta di $\mu\text{C}/51$ venne creata per sviluppare applicazioni di data logging, pertanto caratteristiche come la strutturazione modulare e la presenza dei numeri a virgola mobile erano preponderanti sulla compattezza e la velocità del codice.

La presente versione 1.xx nasce dall'esigenza di creare applicativi web-embedded, pertanto diventano di primaria importanza l'efficienza del compilatore e l'ottimizzazione del codice.

Essa infatti è dotata di un nuovo motore di generazione del codice, molto efficiente, è facile da usare ed è totalmente ANSI compatibile (solo con poche restrizioni dovute alla natura dell'hardware).

PROCESSORI SUPPORTATI

$\mu\text{C}/51$ è **adatto per tutti i membri della famiglia 51**. **Non ci sono requisiti specifici** (come la necessità di RAM esterna). Sono inclusi nella distribuzione molti sorgenti di esempio e file header contenenti la definizione dei registri di numerosi micro 8051 compatibili largamente usati, comprese varie architetture ibride analogico-digitale di notevole interesse (come MSC1210 di TI e ADuC8xx di Analog Devices).

QUALITA' DEL CODICE

μ C/51 è basato su un sistema di modellazione universale, capace di supportare tutti i tipi di CPU ad 8, 16 o 32 bit.

Cosa più importante, questo sistema è stato progettato per architetture 'non lineari', come l'architettura Harvard implementata nella famiglia 51, che separa la memoria del codice da quella dei dati.

μ C/51 è la prima implementazione di questo sistema ed i risultati sono molto buoni. Solo una piccola parte di esso è espressamente dedicata all'architettura 8051, **ma il codice generato può competere facilmente con i leaders di mercato** (basandosi sulle loro descrizioni dei loro prodotti). In alcuni casi il codice è addirittura migliore: ad esempio, μ C/51 traduce il programma dimostrativo incluso nella distribuzione SIEVE (si tratta del classico 'Crivello di Eratostene') in un modulo della dimensione di 142 bytes, il concorrente che più si avvicina ha bisogno del 6% in più. La dimensione totale del codice è 897 bytes, nessuno dei leader di mercato riesce a fare altrettanto.

Se si desiderano ulteriori informazioni sulle tecniche di ottimizzazione, si può consultare il confronto con altri compilatori nell'apposito capitolo di questo manuale.

Se richiesto (come in questo caso) il compilatore utilizza un schema a 'grafo delle chiamate' per minimizzare l'utilizzo delle preziose risorse (la RAM interna, nello specifico della famiglia 51). In combinazione con l'ottimizzazione del flusso dati, la quantità di RAM interna allocata diviene sorprendentemente bassa.

Prima di dare al pubblico la possibilità di usare μ C/51 V1.x sono state realizzate varie **applicazioni di riferimento**, sia come prove interne sia come commesse per clienti:

Un data logger: in grado di controllare una rete RS 485 di sensori. Incorpora un modem GSM (telefonia cellulare), 2 MByte di memoria FLASH (per il codice e i dati), appena 256 Bytes di RAM interna ed un orologio in tempo reale.

Per minimizzare il consumo, il modem si collega solo alla rete cellulare solo per brevi periodi, se necessario è in grado di inviare e ricevere SMS.

Il codice richiede un totale di 15 kByte di memoria e 60 Bytes di RAM interna. Non usa RAM esterna.

Uno stack TCP/IP completo: accedere a **Internet** non è un problema per μ C/51. L'hardware richiesto è disponibile ed è dotato di una interfaccia Ethernet da 10 MB (ulteriori informazioni sulla famiglia **FlexGate**® si possono trovare sul sito www.flexgate.com). La filosofia di FlexGate è di mettere in comunicazione un modulo con una struttura ben conosciuto da una lato, e un vero Web Server dall'altro. Con una minima aggiunta l'applicativo connesso tramite FlexGate è raggiungibile usando un comune Web Browser o la posta elettronica.

Rimane comunque ancora un margine di ottimizzabilità molto ampio, nel futuro verrà implementato tutto il possibile.

LIMITI DELLA VERSIONE DEMO

La versione demo viene fornita con solamente due limitazioni: **non si possono generare più di 8kBytes di codice**. Questa dimensione è più che sufficiente per applicazioni professionali reali. Permette inoltre di usare le routines a virgola mobile e la famiglia di funzioni 'printf()'. Uno dei programmi demo inclusi nella distribuzione implementa una completa analisi di spettro in appena 6kBytes. La seconda limitazione riguarda l'utilizzo: **la versione demo può essere usata solo per valutare il prodotto o per scopi didattici**.

COMPATIBILITA' DEL SORGENTE CON ALTRI COMPILATORI

μ C/51 è totalmente ANSI C. Accetta ogni codice sorgente scritto secondo la specifica ANSI C (con solo alcune piccole restrizioni dovute alla natura della famiglia 51). Vi sono comunque moltissime implementazioni dell'architettura 8051 tutte leggermente diverse e non sempre compatibili tra di loro.

Uno dei più importanti obiettivi del μ C/51 è di offrire un compilatore per codice altamente portabile e facile da usare, e non il solito clone di una implementazione strettamente specifica.

Tuttavia in molti casi il codice sorgente di altri compilatori (per famiglia 51) può essere compilato senza problemi, tenendo semplicemente conto di alcune circostanze. Ulteriori informazioni si possono trovare nell'apposito capitolo del presente manuale o nei commenti dei sorgenti inclusi con la distribuzione.

VERSIONE COMPLETA

La registrazione della versione demo a versione completa viene effettuata via Internet. Si prega di consultare il nostro sito www.grifo.it per ulteriori informazioni.

SUPPORTO

Una licenza μ C/51 è valida per tutte le versioni 'minori' (ovvero V1.xx). Potete scaricare l'ultima versione gratuitamente dal nostro sito. Domande e supporto sull'ultima versione del compilatore sono possibili telefonicamente o eventualmente via e-mail.

INFORMAZIONI SULLA GRIFO®

La **grifo®** è un'azienda appartenente alla categoria PMI situata nel centro-nord dell'Italia. Il nostro indirizzo, numeri di telefono e fax, siti Internet sono riportati in copertina.

PER IL FUTURO

Sono in programma parecchi miglioramenti. Uno riguarda l'interfaccia grafica verso l'utente; sebbene quella attuale sia più che sufficiente, potrebbe essere utile integrarla ed arricchirla di funzioni.

Verranno inoltre aggiunti costantemente sorgenti e headers per le ultime CPU.

TERMINI DI UTILIZZO

Dovete accettare ogni voce di ognuno dei seguenti termini, altrimenti non potete usare μ C/51.

ERRORI

Il software ha subito attente verifiche sia durante lo sviluppo, sia durante le varie prove di idoneità. Tuttavia è ben noto che, visto l'alta complessità delle tecnologie moderne, risulta impossibile garantire, per qualunque prodotto, la totale assenza di errori.

Per questo motivo si declina qualunque responsabilità per danni di qualunque genere casuati direttamente o indirettamente da μ C/51.

INTERNET

Ogni riferimento ad indirizzi internet viene verificato prima del suo inserimento.

Tuttavia non avendo noi alcuna influenza sui contenuti delle pagine da noi citate, ci dissociamo da qualunque contenuto possa essere offensivo di qualunque legge esistente.

INSTALLAZIONE E DISINSTALLAZIONE

μ C/51 funziona con ogni computer 'Win 9x' o superiori. Non ci sono requisiti di sistema particolari. Il software può essere disinstallato completamente dalla routine standard di sistema per la disinstallazione.

I files creati dall'utente, e quindi esclusi dalla disinstallazione del compilatore, possono essere cancellati manualmente.

μ C/51 non effettua alcuna modifica nascosta del vostro computer (come installare DLL nella directory di sistema o creare chiavi non necessarie nei registri di sistema). Viene rimosso al 100% usando il disinstallatore fornito con la distribuzione.

IMPORTANTE: μ C/51 deve essere installato sotto un pathname privo di caratteri spaziatura (ad esempio C:\MieiFiles\uC51\... , ma non C:\Miei Files\uC51\...), perchè l'interprete di MakeFiles non è in grado di gestire pathname contenenti spaziature.

AVVIO

Il μ C/51 è stato pensato come un ambiente flessibile. Per questo è composto da vari pacchetti indipendenti tra loro:

- μ Edit: un editor multi-file facile da usare e dotato di interessanti caratteristiche.
- UmShell: interfaccia utente dell'interprete dei MakeFile di μ C/51. Ha il compito di generare i files binari o Intel Hex.
- FlashMon: downloader per i sistemi che usano OS515 (il cui sorgente è incluso nella distribuzione).
- SLD51: debugger a livello di sorgente (richiede OS515).

INSTALLAZIONE DELLA VERSIONE COMPLETA

Appena installato, μ C/51 è un demo limitato ad 8KBytes di codice per solo scopo valutativo o didattico. Se possedete una regolare licenza per versione completa, dovete prima registrare l'installazione. **Ogni licenza da diritto di installare μ C/51 su due (2) diversi calcolatori**, a patto che il software venga usato su uno solo dei due alla volta. Viene anche fornito un codice di 12 cifre che qui chiameremo 'chiave 1'.

La registrazione avviene in due fasi:

1. Eseguite il programma 'KEY51.EXE'. Se viene trovata una licenza valida sul vostro computer, verrà mostrata, altrimenti dovrete inserire la 'chiave 1'. Fornita questa informazione, KEY51.EXE raccoglierà dati sulla configurazione del computer e li userà per generare una nuova chiave di 20 cifre, chiamata 'chiave 2'. La 'chiave 2' deve essere inviata via e-mail, dove verrà elaborata elettronicamente nel giro di pochi minuti.
2. Se tutto è andato a buon fine, riceverete una risposta alla vostra e-mail contenente il file UC51.KEY. Questo file deve essere copiato nella cartella \BIN della vostra installazione, e il programma KEY51.EXE deve essere avviato nuovamente. A questo punto deve essere trovata una licenza valida.

La riservatezza dei vostri dati personali viene rispettata. La 'chiave 2' contiene solamente informazioni sull'hardware installato nel vostro computer con una elevata dose di ridondanza, quindi potete cambiare molte cose prima che la chiave perda validità.

Se intendete richiedere più di due licenze, potete solamente farlo manualmente con una richiesta all'indirizzo di supporto via e-mail che potete trovare sul nostro sito; in tal caso potrebbe venirci chiesta la motivazione della vostra richiesta.

DOCUMENTAZIONE

La documentazione è complementare ed è distribuita su vari files. Si prega di leggerla attentamente prima di cominciare ad operare. I sorgenti degli esempi sono ricchi di commenti e di informazioni utili. Nel file 'README.TXT' potete trovare le note dell'ultimo minuto.

AVVIO RAPIDO

INSTALLAZIONE

μ C/51 funziona con ogni computer 'Win 9x' o superiori. Non ci sono requisiti di sistema particolari. Il software può essere disinstallato completamente dalla routine standard di sistema per la disinstallazione.

I files creati dall'utente, e quindi esclusi dalla disinstallazione del compilatore, possono essere cancellati manualmente.

μ C/51 non effettua alcuna modifica nascosta del vostro computer (come installare DLL nella directory di sistema o creare chiavi non necessarie nei registri di sistema). Viene rimosso al 100% usando il disinstallatore fornito con la distribuzione.

IMPORTANTE: μ C/51 deve essere installato sotto un pathname privo di caratteri spaziatura (ad esempio C:\MieiFiles\uC51\... , ma non C:\Miei Files\uc51\....), perchè l'interprete di MakeFiles non è in grado di gestire pathname contenenti spaziature.

HARDWARE DI SVILUPPO

μ C/51 è un sistema di sviluppo generico per ogni micro della famiglia 51. Tuttavia per lo sviluppo ed il collaudo dei programmi è conveniente usare una specifica struttura hardware, dove scaricare ed eseguire il codice in RAM.

Naturalmente esistono alternative, come l'emulazione hardware o l'emulazione software.

Viene fornito nella distribuzione un **downloader/debugger** con cui è possibile eseguire passo-passo in **tempo reale** il programma.

Questo software, chiamato **OS535**, è in grado di comunicare con i programmi FLASHMON ed SLD51, forniti nella distribuzione, che ne costituiscono l'interfaccia utente.

OS535 può essere adattato ai diversi micro della famiglia 51 modificando le informazioni di temporizzazione e di baud rate indicate chiaramente dai commenti dentro il sorgente, tuttavia la struttura della memoria è fissa ed è descritta qui di seguito.

ALTERNATIVA

Alcune schede (come FlexGate) offrono due diversi modelli di memoria:

Il **modello uno** permette di scaricare in RAM (come descritto nella seguente figura)

Il **modello due** offre diversi banchi da 64 KByte di memoria FLASH per il codice, la RAM è totalmente libera per i dati. La commutazione tra uno o l'altro modello viene gestita da 'OS535.BIN', in combinazione con un chip di logica programmabile.

Per questo la scheda è da considerarsi come la migliore combinazione sia per lo sviluppo che per l'utilizzo sul campo.

Nel seguito della spiegazione si assumerà l'esistenza di un hardware in grado di eseguire OS535 con la struttura qui indicata (simile a quella di FlexGate).

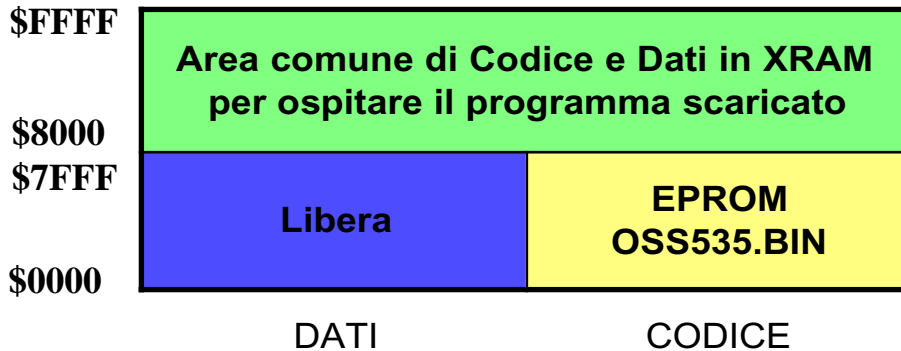


FIGURA 2: STRUTTURA DELLA MEMORIA DELL'OS535

La CPU usata dovrebbe essere un 80c535, c515, 8051 compatibile o un 8051 generico. La maggior parte dei programmi demo richiedono un quarzo da 11, 0592 a 12, 0 MHz. Se usate un 80c535 o un C515 a 12, 0 MHz potete iniziare subito: copiate il file 'src\os\OS535.bin' in una EPROM (il formato del file è binario).

Se usate altre CPU e/o il vostro hardware ha una struttura della memoria diversa, potete adattare il sorgente usando la compilazione condizionale preimpostata o addirittura modificandolo. Basta leggere attentamente i commenti inseriti nel codice sorgente. Dopodiché dovete rigenerare il file binario, come verrà spiegato in seguito.

OS535.BIN si aspetta di trovare una porta seriale (RS 232) ed un LED sul port P3.5 per farlo lampeggiare appena è pronto. Subito dopo il reset il LED lampeggia rapidamente, poi più lentamente. Se si verifica un errore nella trasmissione il lampeggiamento diventa irregolare, quindi bisogna resettare l'hardware.

I demo inclusi per i microcontrollori analogico-digitali funzionano solo sulle loro schede valutative.

HELLO WORLD !

Lanciate l'editor compreso nella distribuzione, uEdit, e digitate:

```
#include <stdio.h>

void main(void) {
    printf("Hello World\n");
}
```

salvate il testo nella vostra directory di lavoro. Non chiudete uEdit.

Adesso lanciate UmShell. UmShell è l'editor dei MakeFiles, ed è basato su un sistema approvato per l'industria chiamato 'Make'. Per un principiante potrebbe sembrare un poco strano, ma di fatto Make è un sistema molto potente e flessibile usato dietro le quinte da molti altri IDE.

Pensate a Make come ad una raccolta di regole su come raggiungere un obiettivo (target).
In questo caso il MakeFile è composto da una sola linea:

```
hello.bin: hello.c
```

Questo istruisce Make sul fatto che hello.bin deve essere ottenuto partendo da hello.c.

Esiste un insieme di regole preimpostato da UmShell (lette da un file) su come generare files di una certa estensione (cioè, per esempio, come generare un file binario *.bin a partire da un file sorgente *.c).

Make inoltre è dotato di macro, una sorta di variabili stringa manipolabili da istruzioni imperative usate, solitamente, per definire i parametri delle linee di comando.

Ad esempio, la macro predefinita 'L51FLAGS' viene usata per notificare impostazioni al linker.

Per quanto riguarda il presente esempio, potete inserir la regola sopra descritta o esaminare come è fatto il file src\hello\hello.mak fornito con la distribuzione. Premendo F9 verrà generato il file, dovrebbe avere una dimensione di circa 300 bytes.

Avviate ora il debugger a livello di sorgente 'SLD51'. Impostate il baud rate appropriato (se diverso da 9600 baud) e scaricate il file hello.bin.

Potete avviare il programma premendo il pulsante col disegno della scarpa. Il programma stamperà all'infinito la stringa 'Hello World' sulla porta seriale. Potete eseguire le istruzioni passo-passo e impostare punti di interruzione (breakpoints) con il mouse.

NOTA: al momento non esiste un modo per oltrepassare un punto di interruzione. Dovete disabilitarlo, superarlo con l'esecuzione di due istruzioni a singolo passo (basta premere <F2> due volte) e riabilitarlo.

Poiché i punti di interruzione sono simulati usando istruzioni 'ljmp' nella RAM esterna, dato che queste istruzioni occupano 3 bytes, non è possibile mettere punti di interruzione ovunque.

Si prevede una futura versione del debugger in grado di lavorare anche con programmi nella FLASH.

IMPORTANTE !!!

Molti dei demo generano files binari (adatti a FLASHMON o SLD51).

Potrebbe essere più efficiente avere un file Intel Hex. Per generare un file Intel Hex aggiungere nel MakeFile la riga:

```
hello.hex: hello.bin
```

Ulteriori informazioni sui MakeFile verranno fornite nei paragrafi successivi.

ANCORA SULLA COMPATIBILITA' ANSI

Ulteriori informazioni sul linguaggio C si possono assumere da libri e corsi. Anche Internet è una fonte preziosa, vi si possono trovare sorgenti ed articoli.

Nel futuro sarà disponibile un manuale didattico per imparare un passo alla volta.

µC/51 è stato progettato nell'ottica della semplicità di uso e della comparibilità con il C ANSI.

Ovviamente, su un microcontrollore non si trovano istruzioni per l'accesso ai dischi rigidi o cose simili. Ma è comunque possibile sviluppare algoritmi su compilatori per PC e poi trasferirli in ambiente embedded se si tiene conto delle risorse più limitate di quest'ultimo.

Gran parte della libreria in virgola mobile e altre cose sono state sviluppate con Borland C++ 5, il sorgente può compilare sia con Borland sia con μ C/51. L'ottimizzatore rende il codice solo leggermente più voluminoso di quello generabile in assembler a mano.

Un altro esempio di codice portabile è il demo 'src\dfht\dfht.c' per cui viene incluso nella distribuzione l'eseguibile per PC.

Consideriamo sia ottimale scrivere funzioni (di libreria) in C e ottimizzare il compilatore per ottenere un livello di qualità 'di compromesso'. Sebbene all'inizio sia più difficile, è conveniente, perchè l'ottimizzatore permette di mettere in campo ciò che si è imparato.

La cartella 'src' contiene svariati programmi dimostrativi, ognuno compilabile mediante il proprio MakeFile.

I commenti nei sorgenti e nei MakeFile sono una lettura sicuramente interessante.

Dentro 'src' l'unico MakeFile ha diversi obiettivi (target) selezionabili con la casella di UmShell.

UN ESAME RAVVICINATO

Come avviene la generazione del file binario a partire dal sorgente C?

Primo Passo: tutti i sorgenti C vengono tradotti in sorgenti assembler (estensione .S51) dal compilatore 'UC51.EXE'. Possono essere necessari file di header, solitamente si trovano nella cartella 'include'.

Immediatamente dopo, l'assemblatore genera un file oggetto (estensione .obj) a partire dal sorgente assembler.

Il primo passo dunque è una traduzione dal sorgente C al file oggetto.

Secondo Passo: il linker elabora tutti i files oggetto insieme per generare il file binario (estensione .bin). Se rimangono riferimenti pendenti, il linker cerca di soddisfarli cercandoli nelle librerie (estensione .lib).

Si possono opzionalmente generare files di simboli (estensione .lst) e/o di mappa della memoria (estensione .map).

Entrambi sono leggibili dall'utente e dal debugger.

DATI FONDAMENTALI DEL μ C/51

BUG REPORT

Se ritenete di avere scoperto un bug, fatecelo sapere. Cercheremo di risolverlo al più presto possibile, per fare questo abbiamo bisogno di un piccolo programma sorgente (non più di circa 20 righe) per riprodurre il problema.

TIPI DI DATI

µC/51 fornisce i seguenti tipi di dati:

char	8 bit (0..255) sono trattati come 'unsigned', compatibile con ANSI
unsigned char	(vedere sopra)
signed char	8 bit (-128..+127) N.B.: una costante di valore -128 è di tipo 'int' (ANSI)
short	16 bit (-32768..+32767) N.B.: una costante di valore -32768 è di tipo 'long' (ANSI)
int	(vedere sopra)
unsigned int	16 bit (0..65535)
long	32 bit (-2147483648..2147483647)
unsigned long	32 bit (0..4294967295)
float	32 bit formato IEEE

char unsigned bit 1 bit (il modificatore 'bit' non è una parola chiave ANSI)

Tutti i dati sono memorizzati nel formato 'big endian'. I tipi non menzionati vengono rimappati all'omologo più vicino (ad esempio: double diventa float, long long diventa long).

Il tipo 'bit' è specifico dell'8051. Può valere 0 oppure non-0 (sebbene la sua rappresentazione come intero sia 1, anche 2135 è non-0...).

PRECISIONE IN VIRGOLA MOBILE

Le routines in virgola mobile sono basate sulle specifiche IEEE. L'arrotondamento viene fatto in conformità a tali specifiche. Tutte le funzioni della libreria matematica (come 'sin()'...) sono state implementate usando algoritmi ed approssimazioni approvate per l'utilizzo in campo industriale.

TIPI DI PUNTATORI

'generico' (far) pointer	32 bit (spiegato nel seguito, può puntare ad ogni cosa)
xdata pointer	16 bit (puntatore alla RAM esterna)
code pointer	16 bit (puntatore alla memoria codice)
near pointer	8 bit (puntatore alla RAM interna dall'indirizzo 0 a 127)
inear pointer	8 bit (puntatore alla RAM interna dall'indirizzo 0 a 255)
bit pointer	<u>non consentito</u>

MODELLI DI MEMORIA

μ C/51 supporta due modelli di memoria: **small** e **large**. Nel modello small tutte le variabili **locali** sono memorizzate nella RAM interna ('near'), quindi questo è il modello più veloce.

Sfortunatamente tale RAM ammonta a solo 128 bytes, quindi se si deve allocare una variabile locale grande bisogna usare il modello large.

Al momento non è possibile mischiare i due modelli nella stessa applicazione. Entrambi usano lo schema 'a grafo di chiamate' per minimizzare l'utilizzo della RAM.

L'unica differenza tra i due modelli è il tipo di memoria usato per allocare le variabili **locali**. Le variabili **globali** possono essere memorizzate ovunque in entrambi i modelli. Di default (se non viene specificato diversamente) queste ultime verranno memorizzate nella RAM esterna o nell'area codice se si tratta di costanti. Usando i modificatori (xdata, code, near, inear e bit) ogni variabile può essere messa individualmente in una qualunque area di memoria. Naturalmente, i puntatori generici possono indirizzare l'intera area di 256 * 16 MB in entrambi i modelli.

Il modello usato di default è small. Per usare il large, ci deve essere l'impostazione 'C51FLAGS=large' nel MakeFile.

NOTA: il modello di memoria non influenza l'allocazione delle variabili globali. Si suggerisce di usare 'typedef' o '#define' per cambiare l'allocazione dei dati globali a seconda del modello usato.

MODIFICATORI SPECIFICI PER 8051

Per distinguere l'allocazione nei diversi tipi di memoria che incorpora ogni micro della famiglia 51, μ C/51 mette a disposizione delle nuove parole chiave (modificatori):

- | | |
|-------|--|
| bit | come 'unsigned char bit', questo tipo di dato occupa solo 1 bit. Ideale per flags globali. Nell'architettura 8051 si hanno 256 bits: 128 sono a disposizione dell'utente mentre i restanti sono vincolati dal microcontrollore per funzioni speciali (SFB). Molti di questi hanno nomi largamente usati, come 'RI'. |
| xdata | questo tipo di dato viene memorizzato nella RAM esterna, per accedervi è necessario usare un puntatore a 16 bit. |
| code | questo tipo di dato viene memorizzato nell'area di memoria del codice, per accedervi è necessario usare un puntatore a 16 bit. |
| near | questo tipo di dato si trova nella RAM interna, negli indirizzi da 0 a 127, vi si accede tramite un puntatore ad 8 bit. |
| inear | questo tipo di dato si trova nella RAM interna, negli indirizzi da 0 a 256, vi si accede tramite un puntatore ad 8 bit. L'utilizzo di questa parte della memoria interna è sempre più lento perché obbligatoriamente indiretto. Risulta essere l'ideale per array globali che devono risiedere in RAM interna. Il vantaggio consiste nel poter comunque usare la parte alta della memoria interna, ove altre strutture, come lo stack, devono risiedere in quella bassa. Quindi, se nel micro che usate la RAM interna non è molta, prendete in considerazione l'uso di questo modificatore. |

far questo tipo di puntatore, detto anche 'generico', può puntare qualunque tipo di dato. Al momento tali puntatori sono solo 'virtuali', potrete dichiarare variabili 'far' a partire da versioni future. In alcuni casi la parole chiave far può essere tralasciata, in tali casi il puntatore diviene 'generico' (analogamente con i tipi di dato 'locali', poiché non esiste un modificatore che li dichiari esplicitamente).

SUI MODIFICATORI, AREE DI MEMORIA E SEGMENTAZIONE

Nei comuni PC non c'è alcun bisogno di modificatori per specificare l'allocazione di memoria, perché solitamente i dati sono accessibili tramite un unico puntatori (normalmente a 32 bit), si dice che il modello di memoria è 'flat', piatto.

Tuttavia non molti anni fa, ai tempi dei software a 16 bit, erano la norma, alcuni sopravvivono, seppure inermi, in alcuni file di intestazione...

Nel $\mu C/51$ i modificatori sono implementati come opzione per programmatori esperti: se non volete progettare l'allocazione di memoria della vostra applicazione, non usateli. Il prezzo da pagare sarà un codice più lento e ingombrante.

Ad esempio, il demo Dhrystone (src\dhry\dhry.c), un classico degli anni passati, non li usa, e tuttavia $\mu C/51$ lo compila ugualmente. Ma l'esecuzione risulta più lenta di quello che potrebbe essere. Alcuni concorrenti offrono modelli di memoria più adatti a demo come questi, ma si rivelano di scarsa utilità pratica, perché hanno diversi svantaggi e sono poco portabili.

Per avere un'idea dell'effetto dei modificatori, iniziamo con un esempio che non li usa.

N. B.: Se non siete interessati all'ottimizzazione spinta nell'uso della memoria, potete saltare questo paragrafo senza problemi, $\mu C/51$ non richiede obbligatoriamente l'uso dei modificatori.

```
int a;
pre_name[]="Bart";
const char last_name[]="Simpson";

void say_hi(char *pc, char *pd){
    printf("Hello %s %s\n",pc, pd);
}

void do_it(void){
    char no[1];
    no[0]=0;
    say_hi(pre_name,last_name);
    say_hi(no,"King Kong");
}
```

'a', 'pre_name' and 'last_name' sono le cosiddette definizioni 'top level', 'a' e 'pre_name' sono variabili globali con accesso in lettura e scrittura, $\mu C/51$ le memorizzerà nella RAM esterna ('xdata'); 'last_name' invece è dichiarata come costante, il compilatore sa che potrà solo essere letta quindi la memorizza nel posto più sicuro, ovvero l'area codice.

Altri compilatori offrono, come detto prima, modelli di memoria ulteriori che impacchettano tutto nella RAM esterna, in modo da poter usare un semplice puntatore a 16 bit per indirizzare tutto, solo che così si ha un incredibile spreco di RAM.

Stesso discorso per la stringa "King Kong", essendo costante verrà memorizzata nell'area codice. La variabile 'no' è locale, non si può scegliere dove posizionarla: con il modello small sarà 'near', con quello large sarà 'xdata'. Anch'essa non è 'top level'.

I puntatori 'pc' e 'pd' potrebbero puntare qualunque cosa nell'area codice dell'8051, non sono 'top level' in quanto puntatori e quindi non vincolati a dati concreti. Quindi il compilatore non è in grado di determinare a quale tipo di dato verranno fatti puntare, per questo divengono puntatori 'generici' a 32 bit. $\mu\text{C}/51$ estende i puntatori automaticamente ad ogni chiamata della funzione `say_hi()`.

Vi siete accorti che ci sono 8 bit di troppo nella dimensione dei puntatori 'generici'? Si ritiene però che il vantaggio apportato da questo byte extra compensi gli svantaggi; in questo modo è possibile accedere a 256 segmenti di tipi diversi di memoria, ognuno lungo fino a 16 MByte!

Ci sono applicativi che fanno già uso di questo byte extra, come FlexGate, che può avere 512 kByte di FLASH o i data loggers da 2 MB.

Vi sono funzioni di libreria a sovrintendere l'accesso tramite puntatori generici. Il sorgente si trova nel file 'lib/lib_ass/mem32.s51'. Siete liberi di aggiungere la vostra gestione della memoria, dopodiché dovete ricompilare le librerie come descritto più avanti in questo manuale.

Ad esempio, in alcuni applicativi di data logging viene mappato ogni dispositivo in memoria: FLASH, RAM interna, persino le periferiche I²C bus.

PER RIASSUMERE

Le costanti intere 'top level' e le costanti stringhe verranno memorizzate nell'area codice, tutti gli altri dati 'top level' che non vengono modificati vanno in xram. Tutti i puntatori non modificati diventano di tipo 'generico'.

Esaminiamo questa definizione globale:

```
char * pc="HELLO";
```

Richiede attenzione: "HELLO" è una stringa costante e quindi andrà nell'area codice. Ma dove andrà 'pc'? Non è una costante, perchè si può cambiarne il valore, quindi $\mu\text{C}/51$ ne farà un puntatore generico. Anche nel caso:

```
const char * pc="HELLO";
```

verrà creato un puntatore 'generico' in area codice.

Per decidere esplicitamente l'allocazione in memoria potete usare i relativi modificatori:

```
xdata int a;           // a è xdata (default)
near char u;          // u è near, accesso rapido!
inchar char x;        // x è inchar, accesso leggermente più lento
inchar buf[50];       // se serve molta memoria interna, inchar è la scelta ottimale
unsigned char bit flag motor_state; // On o off
code float[3]={2.23,3.45,5.99}; // look-up table
```

I modificatori per i puntatori ammettono le stesse regole definite dall'ANSI per i modificatori 'const' e 'volatile':

```
code char* code pc="HELLO"; // pc è un puntatore da 16 bit, situato nell'area codice, che punta
                               // a sua volta nell'area codice
xdata char* near xp;          // xp è 'near' ma i dati a cui punta sono 'xdata'
far char* fp;                 // fp è in 'xdata' (default), 'far' è una specifica obsoleta, ma può
                               // aumentare la leggibilità del codice
```

MODIFICATORI NELLA TYPEDEF

Viene assolutamente sconsigliato! Non ci sono situazioni in cui un modificatore non può in assoluto essere messo fuori da una typedef.

UTILIZZO DEI REGISTRI

Nell'8051 vi sono 4 banchi di registri. Per default $\mu\text{C}/51$ usa i primi due (0 ed 1), ma può essere istruito per utilizzare solo il banco 0, lasciando l'altro a disposizione, ad esempio, di funzioni in interrupt. Un esempio di tale utilizzo si può trovare nel file 'src/misc/soft_rtc.c'. **I banchi 2 e 3 sono completamente liberi.** Vi suggeriamo di riservarli per protocolli, debuggers o altro codice che deve girare in background.

Naturalmente si possono usare per allocare variabili in memoria 'near', ma il linker non è in grado di fare questo automaticamente, perché non è in grado di individuare segmenti 'near' al di sotto dell'area indirizzabile a bit. Quindi il modo più semplice per utilizzare questi (massimo) 16 bytes è:

```
near char buf_a[8] @ 16;
near char buf_b[8] @ 24;
```

Questo crea due buffers nei banchi 2 e 3. La direttiva '@' verrà spiegata in dettaglio più avanti.

INTERRUPTS IN C

Le risposte all'interrupt si possono scrivere in C. $\mu\text{C}/51$ mantiene una lista dei registri corrotti e salverà solo quelli. Per individuare una funzione come risposta ad un interrupt, bisogna precederle il nome con la parola chiave 'interrupt', e deve comunque avere un prototipo.

Per collegare una funzione ad un vettore di interrupt bisogna usare la macro **IRQ_VECTOR**(funcion, cpu_irqaddress) definita nel file '<irq52.h>'. Si veda come esempio il file 'src/misc/soft_rtc.c'.

LAVORARE CON I GRAFI DI CHIAMATA

I grafi di chiamata descrivono la gerarchia logica delle funzioni di un programma. 'main()' è sempre la radice. Con un grafo completo, il linker è in grado di determinare lo spazio totale da usare per le variabili locali.

Ad esempio: se main() chiama due funzioni, a() e b(), entrambe possono condividere lo stesso spazio per le variabili locali ed i parametri (posto che una non chiami mai l'altra).

In ulteriori situazioni potrebbero condividere solo parte dei dati locali, o non dividerli affatto. Questa informazione si può ottenere dal grafo di chiamata.

Grafi di chiamata possono essere totalmente esclusivi o no. Una ragione per avere più di un grafo per ogni programma è la presenza di interrupts o di funzioni C chiamate dall'assembler. Se il grafo non è corretto il linker non è in grado di creare il binario.

La profondità del grafo rappresenta la quantità di stack utilizzata. Nell'8051 ogni chiamata prende 2 bytes, quindi nel grafo ogni chiamata occupa due livelli.

Naturalmente, il compilatore tenterà di usare prima i registri per parametri e variabili locali, solo se servono più bytes di quelli ottenibili in questo modo verrà allocata della memoria.

UTILIZZO DEL FORMATTATORE DELLA PRINTF()

L'output formattato è una risorsa molto preziosa, per questo non è incorporata nella printf(), quest'ultima semplicemente chiama una funzione di formattazione passandole i propri parametri e, in aggiunta, una funzione da usare come gestore dell'output.

Con questo meccanismo diventa molto facile generare output formattato per qualunque periferica. Un esempio si trova nel file 'src\r232_2\r232_c2.c', in cui una funzione di formattazione chiamata 'com2_printf()' viene definita in poche righe.

Nella libreria del µC/51, sprintf() viene implementata proprio usando questo meccanismo.

Altri esempi, l'utilizzo 'trasparente' della memoria di un data logger con un flash_printf() che scriva i dati in FLASH o l'output su un display LCD con le righe non in sequenza usando una lcd_printf().

Il listato completo del formattatore si trova nel file 'lib\lib_c\doprnt.c'. Siete liberi di modificarlo a vostro piacimento. Lo schema consiste nel passare al formattatore una funzione che gestisce l'output di un carattere, la stringa di formato ed i parametri stessi della funzione chiamante.

L'attuale formattatore accetta istruzioni di formattazione del tipo:

% [flags] [dimensione] [prec] [l | L] tipo_dato

[flags] (optionale): -	Giustifica a sinistra il risultato, riempiendo a destra con spazi. Se assente, il risultato è giustificato a destra, con spazi o zeri.
+	Una conversione con segno viene sempre mostrata con un carattere più (+) o un carattere meno (-) iniziali.
spazio	Se il risultato è non-negativo, il valore è stampato con uno spazio iniziale anziché un più, se il risultato è negativo inizia con un meno.

[dimensione] (opzionale):	Numero minimo di caratteri da stampare, da giustificare con zeri o spazi.
[prec] (opzionale):	Numero massimo di caratteri da stampare; per gli interi è il numero minimo di cifre da stampare.
[l L] (opzionale):	Tratta il successivo dato come un intero a 32 bit anziché 16 bit.
tipo_dato:	<ul style="list-style-type: none"> u formato senza segno d formato con segno x stampa in esadecimale, con le lettere minuscole X stampa in esadecimale, con le lettere maiuscole f formato in virgola mobile (tipo [-]dddd.dddd) e formato scientifico (tipo [-]d.dddd e[+/-]ddd) E formato scientifico (tipo [-]d.dddd E[+/-]ddd) g come per e G come per E s stringa % il carattere '%'

come si può vedere, l'elenco dei formattatori è ben fornito.

Il formattatore restituisce sempre il numero di bytes scritti (ovvero il numero di chiamate alla funzione di gestione del singolo carattere).

NOTA: Per l'output verso un UART, ogni carattere newline ('\n') viene sostituito dalla sequenza '\r\n'. Ciò è necessario per la maggior parte degli emulatori di terminale (FLASHMON e SLD51 sono compatibili), la sostituzione viene effettuata direttamente dal gestore dei singoli caratteri per l'I/O seriale (si vedano i files 'lib\lib_ass\seriod.s51' e 'lib\lib_ass\seiop.s51'). Quindi per il formattatore ogni carattere newline conta come un singolo carattere.

NOTA: Non c'è alcun bisogno di convertire esplicitamente i tipi di 8 bit in tipi di 16 bit. Viene fatto automaticamente dal $\mu C/51$, come richiesto dallo standard ANSI. La conversione esplicita, necessaria per molti altri compilatori per famiglia 51, non è ANSI compatibile.

SULLE STRINGHE IN GENERALE

Oltre ai caratteri alfanumerici ed agli altri caratteri stampabili, potete specificare il codice di un carattere usando sequenze di escape esadecimali o ottali. $\mu C/51$ interpreta le sequenze come singoli caratteri, permettendo di specificarne al di fuori del campo degli stampabili (ASCII decimale da 20 a 126). Il formato di una sequenza di escape esadecimale è \x<numero esadecimale>, dove <numero esadecimale> può essere un numero esadecimale fino a due cifre (0÷F). Per esempio, la stringa "R3" si può scrivere come "\x52\x33" oppure "\x52\x33".

Le sequenze ottali sono \ooo, dove ooo sono fino a tre cifre in ottale. Per esempio, "R3" in ottale si può scrivere "\122\x063" oppure "\122\063".

MISCHIARE C ED ASSEMBLER

Mescolare C ed assembler è molto facile con il $\mu\text{C}/51$. Lo stesso compilatore è stato progettato per facilitare l'utilizzo con l'assembler, si possono infatti passare fino ad 8 bytes di parametri tramite registri.

Dopo un reset, il microcontrollore raggiungerà la funzione `main()` mediante un'istruzione di salto 'l`jmp`'. La prima istruzione di `main()` è una chiamata alla funzione `startup()`, che inizializza i dati globali per il programma e lo stack pointer, poi ripartirà con il codice utente. Dopo l'uscita dalla `main()`, tutto riprende dall'inizio.

Quindi chiamando funzioni assembler da C, si troverà comunque l'infrastruttura del programma già inizializzata, si può addirittura ottenere che `startup()` inizializzi i dati delle routines assembler, come viene fatto in alcuni demo inclusi nella distribuzione.

Poiché potreste usare nomi di variabili in conflitto con nomi di risorse interne della CPU (come A o B), si è deciso di anteporre un carattere di sottolineatura ('_') a tutti i simboli esportati dal compilatore.

Ad esempio, una chiamata alla funzione `test()` da C verrà tradotta con un'istruzione 'l`call _test`' in assembler nel file generato come output dal compilatore. Da assembler è possibile accedere a tutte le variabili del C e viceversa, tenendo sempre conto del carattere di sottolineatura all'inizio.

Il passaggio dei parametri tramite registri è molto semplice: tutti i parametri vengono passati usando il banco 0. Questo viene diviso in quattro gruppi di due bytes a partire da R7:R6 (rispettivamente byte alto e byte basso) fino a R1:R0 rispettivamente byte alto e byte basso).

L'allocazione viene fatta da sinistra a destra, iniziando da R7.

Se il primo parametro occupa un byte, andrà in R7, se ne occupa due andrà in R7:R6 (rispettivamente byte alto e byte basso). Il secondo parametro andrà in R5 o in R4:R5 (rispettivamente byte alto e byte basso), oppure, se è di quattro bytes, andrà in R4÷R7 (R4:R5 è la word più significativa) oppure in R3÷R0, a seconda dei registri disponibili.

Il risultato viene passato tramite R7, R7:R6 (rispettivamente byte alto e byte basso) o R4÷R7 (R4:R5 è la word più significativa).

Una funzione assembler può modificare tutti i registri e può usare il banco 1 per le variabili locali se l'accesso al suddetto banco è abilitato, il che corrisponde al default.

Se si passano vari parametri ad una funzione verranno memorizzati nella memoria dedicata ai parametri locali, nel caso dell'assembler questo viene sconsigliato (richiede una modifica al grafo delle chiamate), sarebbe meglio passare alla funzione un puntatore.

UTILIZZO DELL'ASSEMBLER

Si sono due modi:

Il primo consiste nel mettere tutte le funzioni assembler in un file separato (con estensione .s51), come è stato fatto nei demo.

Il secondo consiste nell'incorporare nel sorgente C un blocco di istruzioni assembler dentro le macro #asm / #endasm. Questo comporta il vantaggio di poter usare il preprocessore del C anche con l'assembler (come nel file src\rs232_2\rs232_c".c").

Se c'è un'unica riga assembler si può usare la direttiva #asmline, l'istruzione segue immediatamente (ad esempio #asmline nop) oppure nella riga successiva.

Sebbene sia possibile inserire assembler in una funzione C, questo viene sconsigliato, perché potrebbe interferire con l'attività dell'ottimizzatore, il quale a volte riordina completamente la sequenza di esecuzione delle operazioni.

Un buon metodo potrebbe essere di scrivere lo scheletro delle funzioni in C e lavorare sul file generato dal compilatore.

La sintassi di μ C/51 è leggermente diversa da quella standard: le differenze comunque riguardano solo le direttive assembler e la rappresentazione dei numeri, le parole chiave del linguaggio sono totalmente compatibili con lo standard.

Per ulteriori informazioni si può consultare il capitolo di descrizione tecnica dell'assembler.

ALCUNI SUGGERIMENTI PER USARE L'ASSEMBLER

Blocchi di codice contigui andrebbero memorizzati a segmenti, ogni segmento inizia con una direttiva '.segment <nome segmento>, [<opzioni>]'. I segmenti con lo stesso nome verranno unificati dal linker.

La parte <opzioni> può essere 'sclass <classe>', 'org <origine>', 'size <dim>', 'notext'.

<classe> può essere 'dram', 'iram', 'xram', senza alcuna classe specificata il segmento è codice.

<org> specifica una posizione iniziale assoluta (alla quale il linker può aggiungere un offset se è presente il parametro -r<codice>,<ram>).

<dim> è la dimensione da imporre al segmento.

notext se **non** è presente, un segmento identico con inizializzazione dei dati viene aggiunto nell'area codice (usato dalla funzione di startup per inizializzare i segmenti di dati).

Le etichette di un segmento si possono esportare con una direttiva .export o importare con una direttiva .import. Più avanti ci sarà una descrizione più dettagliata dell'assembler.

FUNZIONI INDIRETTE

Le funzioni indirette si possono usare con una sola limitazione: al massimo 6 bytes per i loro parametri.

Ogni funzione indiretta diventa la radice di un nuovo grafo di chiamata, quindi il linker non condivide la memoria delle loro variabili locali.

FUNZIONI CON NUMERO DI ARGOMENTI VARIABILE

Vengono trattate come nel C standard, si veda a proposito il sorgente della funzione printf(). L'unica eccezione è che gli argomenti vengono sempre promossi ad un intero, dovete tenerne conto negli accessi successivi.

PROMOZIONE DEGLI INTERI

Come da definizione ANSI: tutti i risultati di dimensione inferiore al tipo int vengono estese alla dimensione del tipo int, analogamente i float vengono estesi a double. Nei microcontrollori ad 8 bit questo produce pesanti aggiunte al codice.

Potete abilitare la promozione degli interi anche con $\mu C/51$, che di default è disabilitata, ma il compilatore usa già un algoritmo intelligente: estende la dimensione solo se necessario.

Per le funzioni viene omessa la promozione se è presente un prototipo della funzione.

FUNZIONI OBSOLETE

Si può usare lo stile obsoleto per dichiarare le funzioni, ma ad esse verrà applicata la promozione degli interi anche se esiste un prototipo. Mischiare dichiarazioni obsolete e non può portare a risultati indesiderati.

DEFINIRE UNO SFR, INDIRIZZI ASSOLUTI MEDIANTE '@'

Una delle ragioni del successo della famiglia 51 è che i suoi nuovi membri (successivi cioè all'8051 classico) avevano 'registri speciali' (SFR) più potenti o addirittura nuovi.

Per utilizzare i registri speciali di ogni singolo micro, è presente un file di definizione dei registri, ad esempio il file 'include/reg51.h' o 'include/reg52.h' oppure 'include/reg535.h'.

Ci sono due modi per estendere o modificare le definizioni nei files:

Il primo modo va usato se è molto utile avere disponibili anche per i sorgenti assembler tali definizioni oltre che per il C: prima si aggiunga la definizione valida per l'assembler (ad esempio `WPM_BIT = P3.4`).

Poi, in un'altra riga, mappare la definizione con un simbolo valido per il C, perché il C aggiunge sempre un carattere di sottolineatura davanti ai simboli (quindi `_WPM_BIT = WPM_BIT`).

Infine dare una dichiarazione valida per il compilatore C: `extern unsigned char bit WPM_BIT`.

Il secondo modo è più facile ma il simbolo così dichiarato si può usare solo col C; si aggiunga semplicemente la dichiarazione: `unsigned char WPM_BIT @ 0x85`.

RIDEFINIRE LE FUNZIONI DI LIBRERIA

Potete ridefinire le funzioni della libreria semplicemente aggiungendo al vostro progetto un file oggetto (estensione .obj) che contiene funzioni con lo stesso nome di quelle di libreria che volete ridefinire.

I nomi nei file oggetto del progetto hanno sempre la precedenza su quelli delle librerie.

Se ridefinite un simbolo dell'assembler, il linker userà il vostro, emettendo un avvertimento che potete tranquillamente ignorare.

LA FUNZIONE STARTUP()

La funzione startup() è responsabile dell'inizializzazione dello stack del micro e di tutti i dati in RAM (anche dei bit): tutte le variabili vengono poste a 0 o al valore iniziale stabilito dal programma. Potete personalizzare la funzione startup() e farle fare del lavoro extra a vostro piacimento, si veda in proposito il file 'lib\lib_c' (non è una funzione C ma richiede il preprocessore).

Può capitare di dovere eseguire del codice prima di ogni inizializzazione; un esempio viene fornito per l'impaziente Watch Dog del C515. Dato che ha un timeout di solo 65 msec, e una lunga inizializzazione potrebbe durare di più, il risultato sarebbe che il micro si resetta continuamente senza mai poter terminare l'inizializzazione.

Per ovviare a situazioni simili l'utente può definire una funzione che viene chiamata immediatamente dopo il reset, dato che comunque lo stack e la RAM interna vengono inizializzati questa funzione può essere scritta in C:

```
// Questo comunica al linker di chiamare per prima una funzione dell'utente
#asm
.export START_DEF
START_DEF=1
#endasm
// La funzione con precedenza: Il nome è fisso
void _startup_first(void)
```

Nel caso precedente del C515 una soluzione può essere attivare un interrupt ogni 60 msec che ritardi l'attivazione del Watch Dog a 16 secondi.

Oppure si può eseguire direttamente _main se non serve alcuna inizializzazione della RAM esterna.

CONTROLLO DI INTEGRITA': LA FUNZIONE '_BIN_SAFE()'

Usando questa funzione un programma può controllare l'integrità dei propri bytes, l'uso principale è bloccare ogni modifica al file binario del programma. Ad esempio, se dovete distribuire un aggiornamento di un vostro applicativo in formato binario via posta elettronica, un utente smalzato potrebbe tentare di modificare le schermate senza danneggiare la funzionalità del programma.

_bin_safe() quasi certamente rileverà la modifica, perchè non calcola una semplice checksum sul corpo del programma, ma una calcola un CRC a 16 bit, più difficile da falsificare.

La funzione `_bin_safe()` restituisce 0 se non rileva modifiche, chi crea il programma può decidere cosa fare in caso contrario senza alcuna restrizione. Il prototipo si trova nel file `'_bin_safe()'`; a causa della complessità dell'algoritmo, la funzione non è molto veloce: su un 8051 a 12 MHz si possono esaminare al massimo 8 kBytes al secondo.

EFFICIENZA DEL CODICE

Tenete sempre presente che l'8051 è una CPU ad 8 bit, ove la maggior parte dei libri sul C danno per scontata una CPU a 32 bit. Per l'8051 le operazioni senza segno sono molto più efficienti di quelle col segno, ed il tipo di dato nativo non è `int` ma `unsigned char`.

SIMBOLI PREDEFINITI

Il compilatore definisce alcuni simboli di preprocessore tipici (a parte `__UC__` e `_i8051`):

<code>__DATE__</code>	data odierna (il <code>'__'</code> sono due <code>'_'</code>)
<code>__TIME__</code>	orario corrente
<code>__FILE__</code>	nome del sorgente
<code>__LINE__</code>	numero di linea attuale
<code>__STDC__</code>	definito come <code>"__STDC__"</code>
<code>__UC__</code>	definito come <code>"__UC__"</code> , specifico del μ C/51
<code>_i8051</code>	definito come 1, specifico del μ C/51 (solo un <code>'_'</code>)

OPZIONI IMPORTANTI (LINEA DI COMANDO, #PRAGMA)

Il compilatore può essere controllato sia da linea di comando, sia tramite direttive `#pragma`. Una lista completa delle opzione per la linea di comando si può ottenere lanciando `'UC51 -?'` dalla cartella bin. Qui sono descritte solo le più importanti, si possono passare anche dall'interno di un MakeFile usando la variabile `C51FLAGS`.

<code>-g</code>	Livello di debug (1 (= default) o 2). Con 2, ci sono più informazione ma minore ottimizzazione.
<code>-size</code>	Ottimizzare la dimensione del codice, attivo di default.
<code>-speed</code>	Ottimizzare la velocità del codice.
<code>-a</code>	Sopprimere i messaggi di warning verbosi.
<code>-w</code>	Sopprimere tutti i messaggi di warning.
<code>-dMACRO</code>	Definire una Macro (per assegnare una valore usare <code>'-dMACRO=VALUE'</code>).
<code>-b0</code>	Usare solo i registri del banco 0 (per default verranno usati i banchi 0 e1).
<code>-nograph</code>	Disabilitare l'uso dei grafi di chiamata. Aumenterà molto la RAM necessaria.
<code>-noline</code>	Non generare codice per la macro <code>'__line'</code> . Potete creare la vostra versione della macro, si veda la descrizione tecnica dell'assembler.

Alcuni parametri sono specificabili anche come direttive #pragma:

'#pragma option -gX'	Impostare livello di debug, X=1 o 2.
'#pragma option -a'	Sopprimere i messaggi di warning verbosi.
'#pragma option -a-'	Abilitare i messaggi di warning verbosi.
'#pragma option -w'	Sopprimere tutti i messaggi di warning.
'#pragma option -w-'	Abilitare tutti i messaggi di warning.
'#pragma cpu -b0'	Usare solo banco di registri 0.
'#pragma cpu -b0-'	Usare banchi di registri 0 ed 1.
'#pragma cpu -noline'	Non generare codice per la direttiva .line.
'#pragma cpu -noline-'	Generare codice per la direttiva .line.
'#pragma cpu -nograph'	Non generare inforazioni per i grafi di chiamata.
'#pragma cpu -nograph-'	Generare inforazioni per i grafi di chiamata.
'#pragma cpu -large'	Passare a modello di memoria large.
'#pragma cpu -small'	Passare a modello di memoria small.
'#pragma cpu -labeldist X'	Impostare una nuova distanza tra etichette per i salti brevi.
'#pragma cpu -labeldist-'	Ripristinare la distanza tra etichette.

La distanza tra etichette è un numero, attualmente impostato di default a 50. Poiché $\mu C/51$ non ha un assembler interno, non è in grado di determinare se può ottimizzare un salto mettendolo breve. Per cui tira a indovinare: la maggior parte delle istruzioni sono lunghe meno di due bytes, quindi una distanza di 50 istruzioni tra le due etichette di partenza e di arrivo garantisce che si possono usare con sicurezza i salti brevi. Aumentare il valore potrebbe diminuire la dimensione del codice, a rischio di errori.

UMSHELL E UMAKE

Come già detto in precedenza, questi due programmi determinano la generazione del codice usando una serie di regole voi scriverete nel MakeFile. UmShell è l'interfaccia utente, ma Umake fa la parte più pesante del lavoro.

I FLAGS PIU' IMPORTANTI NEI MAKEFILE

I flags dei MakeFile non sono altro che variabili in grado di memorizzare una stringa. Esse sono:

- C51FLAGS:** contiene i parametri per il compilatore, di default è vuota. Si può usare per passare al compilatore, ad esempio, una definizione di macro del tipo
C51FLAGS=-dTEST -dABC=3, la quale ha lo stesso effetto dell'inserire righe '#define TEST' e '#define ABC 3' nel codice sorgente.
- A51FLAGS:** contiene i parametri per l'assemblatore, di default sono '-d' e '-g'
 -d: espandere la macro __line ogni riga (necessario per l'esecuzione passo-passo)
 -g: includere informazione sul sorgente nel listato

- L51FLAGS:** parametri per il linker, di default '-r\$8000,\$F000', che definiscono come primo byte dell'area codice quello all'indirizzo \$8000 e come primo byte per la RAM esterna quello di indirizzo \$F000.
- MODEL:** Modello di memoria: small di default, definire large se necessario.
- SIOTYPE:** Libreria da usare per l'I/O seriale. Di default vale 'd', che indica di usare la seriale in interrupt. In alcuni casi la libreria gestita in polling potrebbe andare meglio, poiché è in grado di intercettare tutti i valori da 0 a 255 ed occupa meno spazio, ma non permette di fare il debuggin a livello di codice sorgente. Impostare il valore 'p' per usare la libreria in polling.
- PFLAGS:** Se impostato con 'PFLAGS=FULL_PRINTF' viene usato il formattatore completo per la printf(). Questo può essere utile se le stringhe di formato vengono generate dinamicamente durante l'esecuzione. Di default viene usato il valore che permette al compilatore di usare il formattatore minimo che soddisfa le esigenze di ogni programma, 'SMART_PRINTF'.

La definizione di default delle macro appena viste si trova nel file 'bin\builtins.mak'. Se volete ulteriori informazioni sugli altri parametri, avviate ogni singolo programma (compilatore, assembler, linker, ecc.) con il parametro '-?' sulla linea di comando.

Uno degli argomenti più importanti è 'A51FLAGS -d ...': normalmente il compilatore pone una definizione della macro __line all'inizio di ogni programma che traduce, al momento la macro non fa altro che una chiamata all'locazione \$0006, dove dovrebbe risiedere il debugger.

Quindi l'assembler espanderà questa macro per ogni riga C nel proprio output (permettendo il debugging passo-passo a livello di sorgente C).

Nel programma completo, comunque, questa caratteristica andrebbe disabilitata, per non sprecare 3 bytes ogni linea C di codice (il programma non andrà in crash perché comunque verrebbe chiamato un handler di default).

In L51FLAGS potete usare il formato 0x o la '\$' iniziale indifferentemente per esprimere numeri in base 16.

DESCRIZIONE DEI MAKEFILES

Qui segue una descrizione minimale dei MakeFiles.

L'idea alla base è quella di un contenitore di regole che insegnano come ricavare un certo oggetto (obiettivo, goal) a partire da un altro oggetto, se l'obiettivo è meno recente dell'oggetto di partenza. Quindi ogni utility Make fa solo quello che è necessario fare. La regola viene identificata in base all'estensione del nome dei file. Le regole di default si trovano nel file 'bin\builtins.mak' e servono a svolgere i compiti più comuni:

- Creare un file binario (.bin) da sorgenti C (.c) o assembler (.s51)
- Creare un file oggetto (.obj) da sorgenti C (.c) o assembler (.s51)
- Creare un file binario (.bin) da sorgenti oggetto (.obj) o file di libreria (.lib)
- Creare un file Intel hex(.bin) da un file binario (.bin)

Per definire una regola in un MakeFile va sempre specificato prime l'obiettivo, seguito dal carattere di due punti (:), poi segue la lista dei file sorgenti. Per eseguire il lavoro, Make deve poter trovare il nome del file obiettivo in uno dei file sorgente:

```
happy.bin: fast.obj happy.obj newlink.obj
```

Se almeno uno dei tre file sorgente è meno recente di happy.bin, Make tenterà di trovare una regola per rigenerarlo. Trovando dei file oggetto (.obj), userà la regola per generare dei file binari da dei file oggetto. In caso di successo, happy.bin verrà sovrascritto e quindi avrà una data più recente.

La seguente riga, ovviamente:

```
happy.hex: happy.bin
```

genererà il file Intel hex a partire dal file binario.

In alcuni casi la regola non potrà generare il risultato perché, ad esempio, sono stati cambiati i file di header in un sorgente. Make non è in grado di rilevare le dipendenze di questo genere, ma può essere istruito per rigenerare tutto il progetto ignorandole.

Non importa in che posizione viene definita una macro, perché un MakeFile viene sempre letto completamente prima dell'esecuzione. Una macro può essere ridefinita, e anche usata come parametro, semplicemente mettendola tra parentesi e facendo precedere le parentesi dal carattere '\$'. La macro '\$*' equivale al nome del file obiettivo (senza estensione), la macro '\$<' viene sostituita con l'insieme di tutte le dipendenze (con UmShell premere <F8>).

Si prevede di dotare le distribuzioni future di μ C/51 con un gestore di MakeFile grafico, che permetta le abituali di trascinamento e di taglio e incollaggio alle quali siamo stati abituati dalle interfacce grafiche moderne.

REGOLE IMPLICITE E REGOLE ESPLICITE

Umake supporta due tipi di regole: implicite ed esplicite.

Le prime attivano le regole di default citate in precedenza. Ad esempio:

```
.s51.bin:
    $(A51) -e $*.s51 -i$(INCLUDE) $(A51FLAGS)
    $(L51) -e -o$*.bin $*.obj -I$(STDLIB) $(L51FLAGS) -m -s
```

Questa regola di default converte un file assembler (.s51) in un file binario (.bin).

Una regola implicita si definisce designando un obiettivo e alcuni oggetti sorgente; se uno dei nomi dei file sorgenti (senza estensione) coincide, la corrispondente regola di default verrà usata.

Ad esempio:

```
a.bin: a.s51
```

semplicemente convertirà un file assembler in uno binario.

In una regola esplicita invece, vengono forniti esplicitamente dei comandi per scopi particolari. Riprendendo l'esempio precedente, ma volendo usare un assembler-linker diverso, si deve scrivere:

```
a.bin: a.s51
    my_assli -o$*.bin $*.obj
```

CREARE LE PROPRIE REGOLE

Poiché Make è una delle utility più versatili per generare files, è facile espanderne il comportamento aggiungendo le vostre regole.

Immaginate lo scenario seguente: avete appena creato un nuovo traduttore in grado di generare un file assembler a partire da un file HTML dal quale volete facilmente ottenere un file oggetto (potrebbe tornare utile con un Web Server embedded).

Questo traduttore può avere come nome 'html2ass' ed essere invocato passandogli su riga di comando rispettivamente il nome del file sorgente e del file destinazione. La regola implicita potrebbe essere:

```
.htm.obj:
    $(HTML2ASS) -e $*.htm $*.ass #Prima si genera il listato assembly
    $(A51) -e $*.s51 -i$(INCLUDE) $(A51FLAGS) #Poi genera il file oggetto
```

Nell'esempio la macro HTML2ASS contiene il path completo per l'eseguibile 'html2ass.exe'.

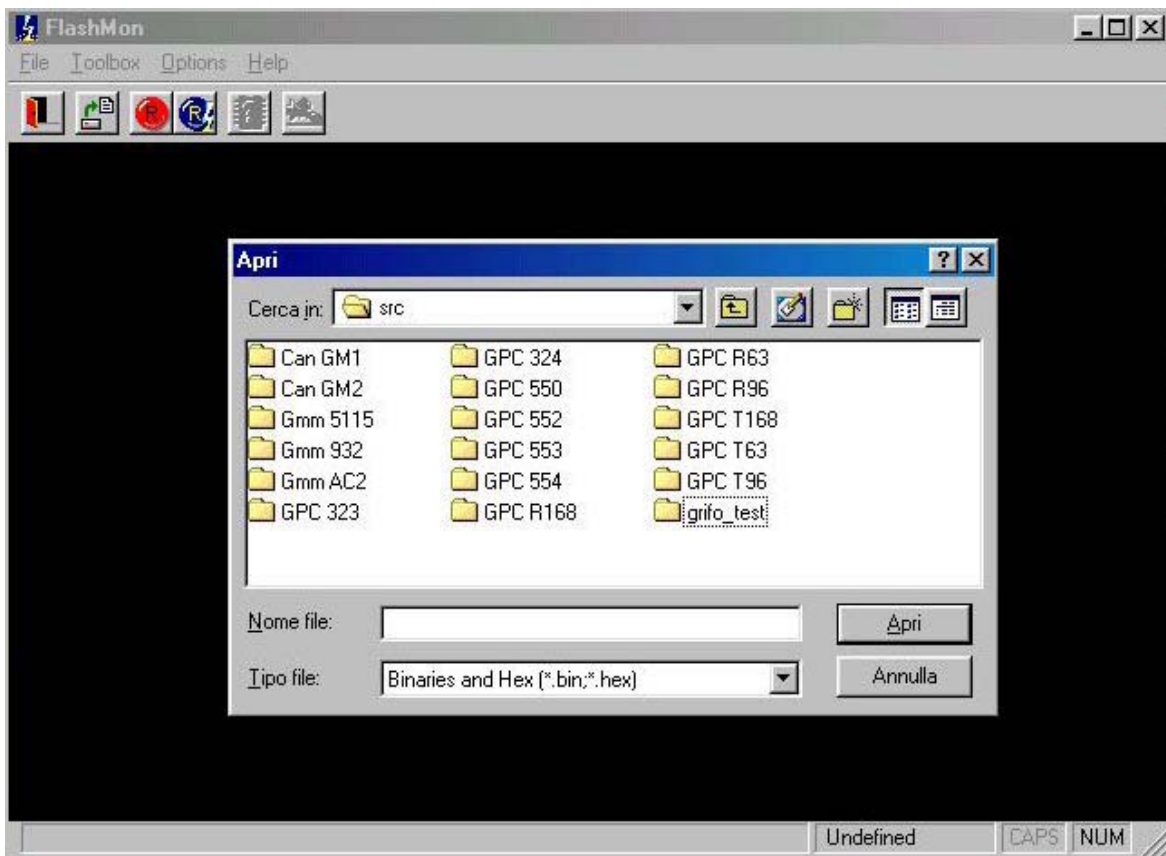


FIGURA 3: IL DOWNLOADER

DESCRIZIONE TECNICA DELL'ASSEMBLER A51.EXE

Il macro-compilatore A51 è stato progettato per lavorare in maniera stabile anche su grossi sorgenti, così come il compilatore. Il suo progetto è indipendente dal tipo di assembly. L'assemblatore opera con una singola passata, per cui è molto veloce.

MNEMONICI

I mnemonici usati sono quelli standard, essi non sono case sensitive, ovvero non distinguono maiuscole e minuscole, ma i simboli che potete definire sono case sensitive.

Ad esempio: `mov acc,#0` si riferisce al registro accumulatore, `mov ACC,#0` si riferisce al simbolo 'ACC'.

Per convenzione tutti gli SFR sono in maiuscolo.

Ovunque si debba usare una costante numerica, questa può essere calcolata. Ad esempio:

```
WORKREG=5  
mov R(WORKREG+1), a
```

equivale a:

```
mov R6,a.
```

NOMI, VARIABILI, ETICHETTE

Tutti i nomi di variabili seguono le stesse regole del C: il primo carattere deve essere '_' oppure una lettera maiuscola o minuscola, gli altri carattere possono anche essere numerici.

Un nome non può essere più lungo di 64 caratteri.

Le etichette terminano con un carattere di due punti ':' e sono trattate come indirizzi.

Una variabile è un simbolo che può contenere un valore. Tale valore può essere riassegnato:

```
nr=11  
mov a,#nr ;nr vale 11  
nr=21  
mov a,#nr ;nr vale 21
```

NUMERI

L'assemblatore accetta numeri specificati in vari formati:

123	decimale
'X'	numero di 8 bit (carattere ASCII)
'AB'	numero di 16 bit (come per carattere ASCII: ('A'*256)+'B'))
0x100	numero esadecimale (256 dec.) sintassi del C
\$100	numero esadecimale (256 dec.) sintassi Motorola
0100h	numero esadecimale (256 dec.) sintassi Intel (versione 1)
0100H	numero esadecimale (256 dec.) sintassi Intel (versione 1)
100h	numero esadecimale (256 dec.) sintassi Intel (versione 2)
100H	numero esadecimale (256 dec.) sintassi Intel (versione 2)
0100	numero ottale (83 dec.) sintassi del C
%111	numero binario (7 dec.)
%000111	numero binario (7 dec.)
0111b	numero binario (7 dec.) sintassi Intel (versione 1)
0111B	numero binario (7 dec.) sintassi Intel (versione 1)
111b	numero binario (7 dec.) sintassi Intel (versione 2)
111B	numero binario (7 dec.) sintassi Intel (versione 2)

I formati con \$ e % sono i più leggibili, pertanto sono consigliati.

OPERATORI

A51 utilizza gli stessi operatori del C (a parte l'indirizzamento a bit):

()	Parentesi, aumenta la priorità
.	Indirizzamento a bit (come acc.7: bit più significativo di acc)
* / %	Moltiplicazione, divisione, modulo
+ -	Forza il segno
<< >>	Operatori di shift
<> == <=	Confronto, vale 1 se vero, 0 se falso
&	And bit a bit
^	Or esclusivo bit a bit
	Or bit a bit
&&	And logico
	Or logico

DIRETTIVE

.segment

Utilizzo: `.segment NOME [, parametro]`

Apri o crea un segmento chiamandolo NOME. Il linker raggruppa i segmenti con lo stesso NOME e li tratta come un'unica sequenza di istruzioni. Il compilatore genera un segmento per ogni funzione. Si consiglia di chiamare il segmento come la funzione o la label di ingresso aggiungendo un carattere di sottolineatura ('_').

In quanto alle funzioni, ognuna dovrebbe avere il proprio segmento, proprio come i dati dello stesso tipo dovrebbero essere raggruppati ognuno in un segmento specifico.

Una volta assegnati i nomi dei segmenti per le variabili, la funzione `startup()` può iniziarli correttamente. Il driver seriale in interrupt (file `lib_ass\seriod.s51`) usa questa tecnica per dichiarare una variabile nel segmento 'nearbss', la startup lo inizializza azzerandolo.

Su richiesta il linker può generare simboli aggiuntivi per raggiungere i dati interni dei segmenti (come la dimensione, l'indirizzo di caricamento, etc.), normalmente questa caratteristica non viene usata.

NOTA: Il linker rimuove tutti i segmenti non utilizzati. Rimangono solo i segmenti ai quali viene fatto un riferimento e che contengono una direttiva 'org'.

Il parametro opzionale può essere uno o più tra i seguenti:

org NUMERO

Questo imposta un indirizzo assoluto per il segmento, solo uno per ogni segmento può essere specificato.

Il programma inizia l'esecuzione alla prima etichetta del segmento con `org` pari a 0. Il linker può sommare uno scostamento su richiesta, di default nei MakeFile lo scostamento è \$8000, per l'utilizzo immediato con `SLD51`.

sclass NAME

Imposta la classe di memoria del segmento, se non specificato la classe è automaticamente 'text'.

I nomi utilizzabili sono:

text	Codice e costanti (EPROM). Viene usato come default per i segmenti.
xram	RAM esterna (accesso tramite MOVX).
bit	I bit della RAM interna (da \$20 a \$2F).
dram	RAM interna accessibile direttamente (da \$0 a \$7F).
dram	RAM interna accessibili indirettamente (da \$0 a \$FF).

size NUMERO

Imposta la dimensione massima del segmento. In caso il segmento sia scaglionato in più direttive viene usato il valore più elevato. Il compilatore fa largo uso di questa direttiva per gestire le variabili locali.

fill

Se impostato, il linker imporrà sempre al segmento la sua dimensione massima (va usato insieme al parametro 'size').

notext

Normalmente il linker non ha alcuna possibilità di inizializzare segmenti che non siano di codice (come la RAM). Quindi il linker mette una immagine dei segmenti di dati inizializzati in segmenti particolari detti 'mirror', che possono essere copiati pedissequamente in RAM per ottenere i dati inizializzati.

Questi segmenti sono indicati con il parametro 'notext'.

page NUMERO

Se il parametro è presente, il segmento verrà posizionato dentro la pagina di NUMERO specificato.

Ad esempio: se il codice del segmento usa istruzioni 'acall' e 'ajmp', NUMERO deve essere 2048.

Può essere usato un numero fisso per il segmento (parametro 'size'). Normalmente il valore di NUMERO è una potenza di 2.

align NUMERO

Questo parametro condiziona il primo byte del segmento.

Ad esempio, se il segmento deve iniziare ad un indirizzo pari, NUMERO deve essere 2. Le direttive 'align' e 'page' possono essere usate in qualunque combinazione.

.include

Ammette due utilizzi: '.include <FILE>' e '.include "FILE"'.

Entrambe le versioni inseriscono un listato assembler (file di testo) nel file in cui si trovano.

Differenza è che <FILE> utilizza il path passato come parametro in linea di comando (usando -i) all'invocazione dell'assembler per cercare FILE, mentre con "FILE" questo viene cercato solo nella cartella corrente.

Di default il path degli include assembler è la cartella 'include' del æC/51.

.ibytes

Utilizzo: .iname "FILE"

Inserisce un file binario (ad esempio una tabella dati). Vale solo il formato "FILE".

.error

Utilizzo: .error [TESTO]

Viene indicato un errore e viene arrestata la generazione del file binario. Se presente, viene stampato anche il TESTO.

.end

Viene arrestata la generazione del file binario, ogni testo successivo viene ignorato. Funziona anche da un file incluso.

.import

Utilizzo: .import SIMBOLO [, SIMBOLO, SIMBOLO, ...]

Per accedere ai simboli definiti in altri sorgenti, essi devono essere importati con questa direttiva.

.export

Utilizzo: `.export SIMBOLO [, SIMBOLO, SIMBOLO, ...]`

Esporta uno o più simboli per permetterne l'utilizzo globale. Se un simbolo compare in una `'import'` ed una `'export'` dentro lo stesso sorgente, prevale la `'export'`. Si possono anche esportare assegnamenti fatti in $\mu C/51$ (ad esempio per utilizzare la funzione `_startup_first()`). Questo è utile per le librerie. Ad esempio, se una libreria I2C bus importasse i simboli SDA e SCL senza definirli al proprio interno, una volta inclusa in un progetto il programmatore potrebbe scrivere le seguenti istruzioni nell'inizializzazione del proprio progetto:

```
.export SDA, SCL
SDA=P1.7
SCL=P1.6
```

ottenendo così di associare i simboli ai pin specificati solo per quel progetto.

In un altro progetto i pin potrebbero cambiare senza alcun bisogno di cambiare la libreria.

NOTA: Tenete sempre presente che i simboli generati dal compilatore hanno un carattere di sottolineatura ('_') aggiunto all'inizio, così, ad esempio, la funzione `test()` viene vista dall'assembler come il simbolo `_test`.

.file

Utilizzo: `.file "FILE"`

Comunica all'assembler che il sorgente ad alto livello contenuto in FILE corrisponde al codice assembler del file in cui compare la direttiva.

.line

Utilizzo: `.line NUMERO`

Indica all'assembler che il codice seguente alla direttiva corrisponde alla linea di NUMERO specificato del file indicato con la direttiva `'file'`.

Se l'assembler è stato istruito di produrre un listing (con `-s`) o di includere linee del sorgente ad alto livello nel file oggetto (con `-g`), le linee verranno incluse, altrimenti non succederà niente.

Questa direttiva ha anche un'altra funzione molto importante: disseminare i punti di interruzione (breakpoints) nel file binario.

Se l'assembler è stato istruito di espandere le macro `'__line'`, nel punto in cui una di esse viene chiamata (con un argomento numerico che, al momento, viene ignorato) un'istruzione di salto `'ljmp $0006'` verrà scritta.

All'indirizzo `$0006` può trovarsi il monitor `OS525.bin` oppure una semplice istruzione `'ret'` aggiunta dal file `'lib\lib_c\startup.c'` per prudenza.

Quindi il monitor `OS525.bin` viene chiamato dopo ogni riga (istruzione) del sorgente C.

Questa caratteristica è molto utile per debuggare il software. Sfortunatamente ogni breakpoint occupa 3 bytes di codice extra e produce un leggero rallentamento.

Per disabilitarla, bisogna escludere l'opzione -g dalla chiamata in linea di comando dell'assembler, ad esempio ridefinendo la variabile A51FLAGS nel MakeFile (tramite UmShell) al valore -d (solo listing) o a qualche altro valore. Il default è 'A51FLAGS=-d -g' (nel file 'bin\builtin.mak').

NOTA: Viene studiata al momento la possibilità di creare una nuova versione di SLD51 in cui l'istruzione di salto viene sostituita con qualcosa di diverso.

Inoltre, dato che molti micro della famiglia 51 sono forniti di FLASH interna, e dato che questo metodo non può essere utilizzato con la FLASH, le prossime versioni useranno un sistema che non modifica il codice e che si potrà usare direttamente sul campo.

NOTA SULL'OTTIMIZZAZIONE: A volte il compilatore riordina completamente il flusso logico delle istruzioni di una funzione per ottimizzarla. Imporre il breakpoint potrebbe modificare leggermente la sincronia dell'output. Questo non è un errore.

.macro / .endmacro

Utilizzo: `.macro NOME`

Crea una macro. La macro è una sostituzione del testo con caratteristiche ulteriori. Si veda più avanti.

.if / .else / .endif

Utilizzo: `.if CONDIZIONE`

Usate per la compilazione condizionale; la parte '.else' è opzionale.

La condizione deve essere completamente valutabile nel momento della compilazione.

Non si possono usare etichette nella condizione.

Le direttive si possono annidare, ad esempio:

```
.if DEBUG_LEVEL == 1
    .if SUBVERSION >=2
        lcall test12p
    .else
        lcall test11
    .endif
.endif
```

.ifdef / .ifndef

Utilizzo: `.ifdef SIMBOLO` `.ifndef SIMBOLO`

Verifica se un simbolo è già stato definito, è molto simile alla precedente.

Ad esempio:

```
.ifndef char_out ; Se la macro non è già stata definita
    .macro char_out ; definiscila!
        mov A,@1 ; parametro @1 in A
        lcall output ; output
    .endmacro ; ora la macro è definita
.endif
```

.hide / .show

Aumentano e diminuiscono il livello di documentazione da inserire nel file oggetto.

Solo se il livello è maggiore di zero viene generato il listing e le informazioni del sorgente vengono incluse nell'oggetto.

Durante l'espansione di una macro il livello viene temporaneamente decrementato, quindi se volete informazioni di debugging anche sulle macro, aggiungete una direttiva `.show`. Il livello di default è 1.

.dc.b / .dc.w / .dc.l / .dc.f

Inseriscono numeri e stringhe nel file binario. Più elementi nella stessa riga devono essere separati da una virgola, con `.dc.b` si possono definire delle stringhe. L'ordine dei bytes usato da `.dc.w` e `.dc.l` è Big Endian, cioè viene prima il byte più significativo. Per inserire un numero a virgola mobile a 32 bit IEEE32 si usa `.dc.l`. Ad esempio:

```
.dc.b "Hello World",13,10,0 ; Testo, <CR>, <LF>, 0
.dc.w init, $0000, 'AB' ; 'AB' è $41,$42 (stesso effetto di .dc.b 'A','B')
.dc.w test ; Se test=$8023, ha lo stesso effetto di .dc.b $80, $23
.dc.l init, 'ABCD' ; Costante di 4 byte
.dc.f 3.14159, 2.718282 ; Due numeri a virgola mobile
```

.ds.b / .ds.w / .ds.l / .ds.f

Utilizzo: `.ds.X NUMERO`

Queste direttive non fanno altro che **riservare lo spazio** necessario per il NUMERO specificato di oggetti del tipo indicato, ovvero bytes, words, longs o floats. Lo spazio viene inizializzato con bytes del valore di 255. NUMERO può anche essere 0, in questo caso non viene scritto nulla.

MACRO

Una macro ha l'effetto di una sostituzione testuale con alcune caratteristiche aggiuntive. Una macro può contenere altre macro e utilizzare etichette temporanee.

Prima di essere usata, una macro deve essere definita mediante un blocco **'macro'** / **'endmacro'**.

Ogni macro può avere fino a 10 parametri, utilizzabili con i metasimboli da '@0' a '@9'. Il numero dei parametri forniti si trova in '@0', il primo parametro in '@1'. Un parametro non definito non può essere utilizzato, altrimenti verrà generato un errore. Il parametro di una macro può anche essere una intera stringa.

NOTA: durante l'espansione di una macro, il livello di documentazione viene decrementato.

Vengono generate etichette temporanee ad ogni espansione. Ad esempio:

```
.macro help_text ; Stringhe di help in una tabella
.dc.b @1,@2,0 ; La struttura : Help-ID, Stringa, 0
.endmacro
...
.segment help_tab, sclass text
help_text 22,"Valve open" ; inserisce testo
help_text 25,"Temperature low" ; inserisce altro testo
help_text 17, "(C) Copyright 2003"
```

Oppure:

```
.macro safe ; salva aree di memoria sullo stack
    mov R0,@1 ; start
    mov R1,@2 ; lunghezza
?s0: mov A,@R0 ; non è una parametro della macro, ma indirizzamento indiretto
    inc R0 ; indirizzo successivo
    push ACC
    djnz R1,?s0 ; '?s0' trattato come '?ID_s0' (ID: 1,2,...)
.endmacro
...
safe _temp,17 ; 17 bytes messi al sicuro dall'indirizzo contenuto in _temp
...
safe _temp,10 ; 10 Bytes safe messi al sicuro dall'indirizzo contenuto in _temp
```

In entrambi i casi l'espansione dell'etichetta temporanea '?s0' è diversa, genera un simbolo diverso.

SIMBOLI GENERATI

Su richiesta (se importati con `.import`) l'assembler ed il linker genereranno simboli aggiuntivi, contenenti informazioni sui segmenti. Se '%s' è il nome del segmento, sono disponibili i seguenti simboli, attenzione al doppio carattere di sottolineatura ('__') all'inizio e alla fine:

<code>__%s_org__</code>	indirizzo (reale) di inizio del segmento
<code>__%s_size__</code>	dimensione (reale) del segmento in bytes (in bit per 'sclass bit')
<code>__%s_maxsize__</code>	massima dimensione per il segmento (se impostata)
<code>__%s_load__</code>	indirizzo dell'inizializzazione dei dati nello spazio codice (se impostato)

In aggiunta per ogni 'sclass' ('%s' indica la classe del segmento, ovvero text, bit, dram, iram o xram):

`__%s_org__`
`__%s_size__`

Ed inoltre:

<code>__stack_org__</code>	il primo byte non utilizzato nella RAM interna viene usato per lo stack. Cresce verso l'alto.
<code>__bin_size__</code>	indica la dimensione totale del file binario, compreso tutto il codice e l'inizializzazione dei dati.

LIBRERIE STANDARD

STDIO.H

Uno dei file header più importanti. Tutte le funzioni si comportano come specificato dallo standard. Le seguenti definizioni semplificano l'utilizzo dei tipi di dato nativi della famiglia 51 e risultano quindi molto comode:

```
typedef unsigned int uint;  

typedef unsigned char uchar;
```

```
void putc(uchar) reentrant; // rientrante perché chiamata indirettamente  

uchar getc(void);  

uchar kbit(void);
```

```
// printf() e sprintf() restituiscono il numero di caratteri stampati (ANSI)  

// per ulteriori informazioni sulle stringhe di formato si vedano i capitoli precedenti...  

int printf(far char* pfmt, ... ); // stampa usando la putc()  

int sprintf(far char *dest far char* pfmt, ... ); // stampa in una stringa  

int puts(far char* ps); // restituisce sempre 1 (ANSI)
```

Le seguenti funzioni implementano un generatore di numeri pseudo casuale, basato su un algoritmo lineare congruente. La sequenza di valori generati è sempre la stessa e dipende da un seme. Una variabile globale di 4 bytes viene utilizzata per memorizzare l'ultimo risultato (si veda il sorgente in 'lib\lib_c\rand.c').

```
unsigned int rand(void); // Genera un valore pseudo casuale tra 0..65535  

void srand(unsigned int seed); // Imposta il valore iniziale della sequenza
```

NOTA: Nel caso di un microcontrollore un registro a scorrimento o il valore di uno dei timer potrebbero fare un lavoro migliore.

Funzioni di conversione per stringhe o numeri interi long:

```
int atoi(far char* pc);  
long int atol(far char* pc);
```

Le funzioni qui di seguito non sono standard ma sono spesso utili. La 'puts1()' funziona come la 'puts()', tranne che non aggiunge il ritorno di carrello alla fine della stringa da stampare. La 'inputse()' può essere usata per leggere input da una stringa inserita dall'utente, puntata da 'pc', di dimensione 'max'. Come risultato ritorna la lunghezza dell'input utente (al massimo vale 'max'-1). La stringa deve essere terminata col carattere nullo '\0'.

```
void puts1(far char *ps); // Non Standard  
unsigned char inputse(far char *pc,unsigned char max); // Non Standard
```

NOTA: La funzione inputse() viene usata nel demo 'src\mini535\m535v3.c'.

STRING.H

Solo disponibili le funzioni ANSI fondamentali per la manipolazione delle stringhe:

```
int strlen(far char* px);  
int strcmp(far char* pa,far char* pb);  
int memcmp(far char* pa,far char* pb, int n);  
far char* strcpy(far char * pdst, far char * psrc);
```

Una funzione per spostare bytes non ANSI (notare che l'area 'sorgente' di dati è il primo argomento) ma comunque molto comune:

```
void bmove(far void * _pscr, far void * _pdest, unsigned int count);
```

NOTA: tutte le funzioni per lo spostamento dei bytes usano puntatori generici, in modo da poter spostare bytes da una qualunque area di memoria ad una qualunque altra. Se sono noti a priori i tipi di memoria delle aree di partenza e di destinazione, risulta più conveniente usare una funzione specializzata.

CTYPE.H

```
char tolower(char c);  
char toupper(char c);
```

STDARG.H

Permetton di usare funzioni con numero variabile di parametri secondo lo standard ANSI:

```
va_start(y,y);  
va_arg(x,y);  
va_end(x);
```

BIN_SAFE.H

Viene definita un'unica funzione che verifica l'integrità sul campo del file binario, come già spiegato nei paragrafi precedenti:

```
unsigned char _bin_safe(void); // Se l'integrità è verificata restituisce 0
```

MATH.H

Ogni funzione matematica ha una precisione di almeno 7 cifre significative, ogni algoritmo ha l'approvazione per l'uso industriale.

Costanti:

```
M_PI_2      1.570796326794895 // ANSI  
M_PI       3.14159265358979 // ANSI  
M_TWO_PI  6.28318530717958 // NONANSI
```

Funzioni (ognuna rispetta lo standard ANSI):

```
float atof(far char* pc); // converte una stringa in un numero a virgola mobile  
float sqrt(float f); // radice quadrate  
float sin(float x); // funzione sin() in radianti (standard)  
float cos(float x); // funzione cos()  
float log(float x); // logaritmo naturale  
float log10(float x); // logaritmo decimale  
float exp(float x); // funzione esponenziale  
float pow(float x); // elevamento a potenza
```

HEADER FILES SPECIFICI DELLA FAMIGLIA 51

IRQ52.H

Questo header file definisce delle macro per legare un interrupt ad una funzione di risposta. Sia 'nome' il nome della funzione, 'loc' l'indirizzo o il nome simbolico dell'interrupt (tipo INTO, ecc.):

```
IRQ_VECTOR(nome, loc);
```

REG51.H, REG52.H, REG535.H

Questi header files definiscono gli SFR (si veda nei paragrafi sull'accesso alla memoria) specifici delle CPU di cui portano il nome.

Gli header specifici delle CPU ad architettura mista analogica-digitale si trovano nelle cartelle che ne contengono i relativi esempi.

KAR.H

Includere questo header se si usa il sorgente di altri compilatori.

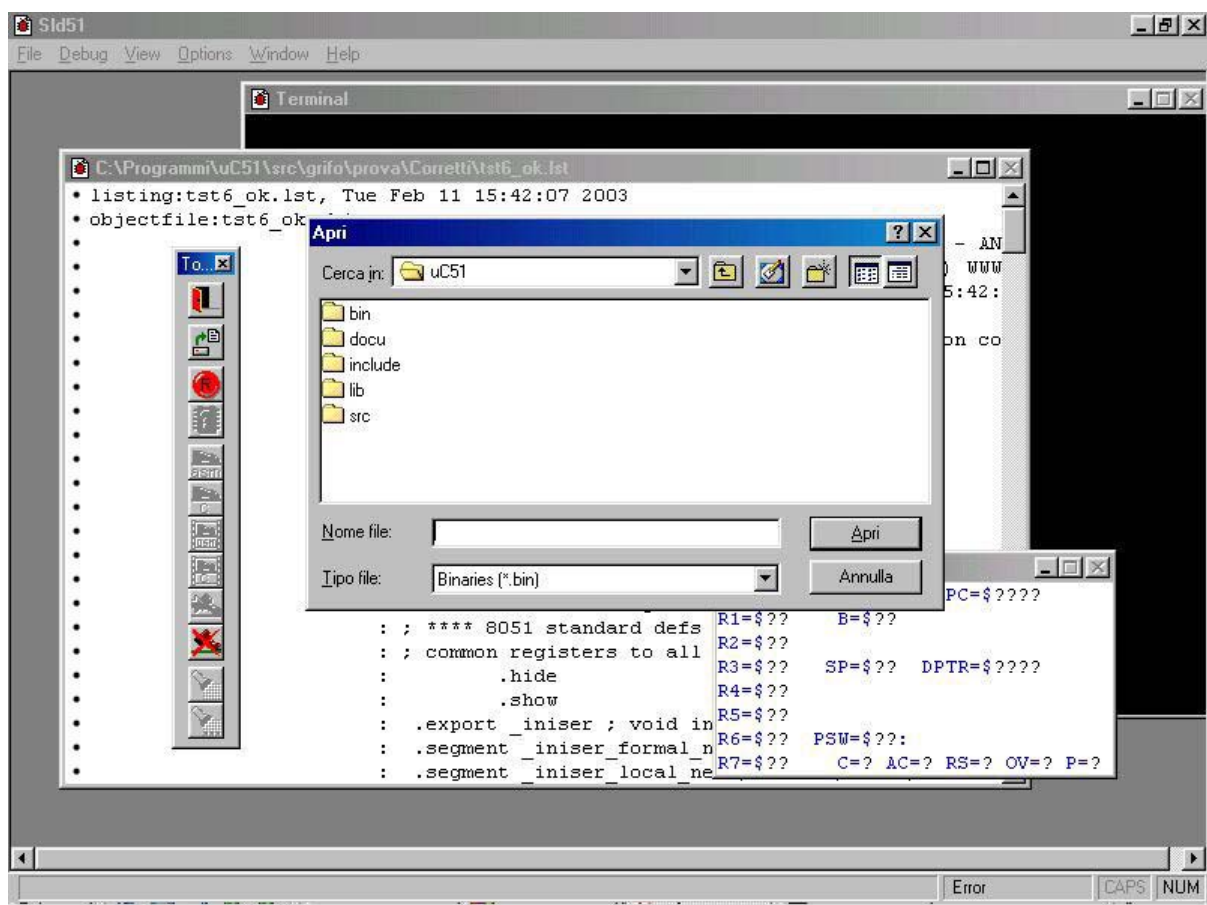


FIGURA 4: IL DEBUGGER A LIVELLO DI SORGENTE

PORTARE DA ALTRI COMPILATORI

Come già detto, $\mu\text{C}/51$ è stato progettato per essere il più possibile conforme allo standard ANSI fermo restando le caratteristiche specifiche per i microcontrollori della famiglia 51.

Per questo la maggior parte dei problemi di conversione da altri compilatori riguarda le estensioni non standard dei suddetti.

Normalmente lo stesso $\mu\text{C}/51$ segnala cosa non va, ma alcune cose vanno tenute presenti:

UTILIZZO DELLA MEMORIA - MODELLI DI MEMORIA

$\mu\text{C}/51$ offre due soli modelli di memoria: 'small' (di default) e 'large'.

L'unica differenza tra i due modelli è il tipo di memoria usato per le variabili locali: RAM interna per il modello 'small', RAM esterna per 'large'.

Le variabili globali non costanti andranno sempre in RAM esterna, a meno di non specificare un'altra destinazione tramite gli appositi specificatori (vedere l'apposito paragrafo per ulteriori informazioni).

Costanti globali vanno in memoria codice.

Le variabili bit sono dichiarate con 'unsigned char bit'.

HEADER FILE UTILI

Il file header kar.h può essere utile se si usano sorgenti da altri compilatori. Contiene alcune definizioni usate dai compilatori leader del mercato mondiale:

```
#define sfr      near unsigned char    // richiede un indirizzo assoluto
#define sbit    unsigned char bit     // RAM indirizzabile a bit
#define data    near                  // RAM interna bassa (128 bytes)
#define idata   inear                  // RAM interna indiretta (256 bytes)
#define bit     unsigned char bit
```

INDIRIZZI ASSOLUTI

Per assegnare un indirizzo assoluto ad una variabile si usa l'operatore '@', ad esempio:

```
near unsigned char P0 @ 0x80;
```

Dopo l'operatore '@' sono permessi solo un numero, un'espressione costante o una costante del preprocessore, ad esempio:

```
#define SCL 0xB0+7
unsigned char bit scl_bit @ SCL;
```


INTERRUPTS

Una procedura di risposta ad un interrupt può essere scritta in $\mu C/51$.

Il vettore non viene compreso nella dichiarazione.

Per legare un interrupt alla funzione di risposta si usa la macro:

IRQ_VECTOR(nome, loc)

come precedentemente spiegato.

LINGUAGGIO ASSEMBLY

L'assembler del $\mu C/51$ usa direttive proprie.

Comunque, i mnemonici sono totalmente compatibili con quelli dell'8051 standard.

Per cui è possibile, e sicuramente conveniente, effettuare a mano la conversione da un qualunque altro assembly a quello del $\mu C/51$, effettuando pazientemente la sostituzione delle direttive che risultano incompatibili.

DIMOSTRAZIONE DELL'OTTIMIZZATORE

Viene qui fatto un confronto tra il compilatore KXXX V6.21 e il $\mu C/51$.

Potete facilmente verificare i dati qui presentati, loro offrono una versione demo limitata a 2 kBytes di codice.

La funzione di test è una qualche implementazione di "bit banger": sposta dei bit a destra e a sinistra.

Ecco il risultato del compilatore KXXX V6.21, richiede 35 bytes per risolvere il problema:

```
bit out_bit ;
bit in_bit ;

// Simple 8-Bit-Banger
unsigned char test(unsigned char ob){
    int i;
    unsigned char ib;
    for(i=0;i<8;i++){
        out_bit=(ob&128);
        ib<<=1;
        if(in_bit) ib|=1;
        ob<<=1;
    }
    return ib;
}
```

```
C51 COMPILER V6.21 BANG
ASSEMBLY LISTING OF GENERATED OBJECT CODE
                ; FUNCTION _bang (BEGIN)
;---- Variable 'ib' assigned to Register 'R6' ----
;---- Variable 'ob' assigned to Register 'R7' ----
;---- Variable 'i' assigned to Register 'R2/R3' ----
0000 E4 CLR A
0001 FB MOV R3,A
0002 FA MOV R2,A
0003 ?C0001:
0003 EF MOV A,R7
0004 33 RLC A
0005 9200 R MOV out_bit,C
0007 EE MOV A,R6
0008 25E0 ADD A,ACC
000A FE MOV R6,A
000B 300003 R JNB in_bit,?C0004
000E 430601 ORL AR6,#01H
0011 ?C0004:
0011 EF MOV A,R7
0012 25E0 ADD A,ACC
0014 FF MOV R7,A
0015 0B INC R3
```

```

0016 BB0001 CJNE R3,#00H,?C0009
0019 0A INC R2
001A ?C0009:
001A EB MOV A,R3
001B 6408 XRL A,#08H
001D 4A ORL A,R2
001E 70E3 JNZ ?C0001
0020 ?C0002:
0020 AF06 MOV R7,AR6
0022 ?C0005:
0022 22 RET

```

```

; FUNCTION _bang (END)

```

Ed ecco la soluzione del μ C/51, richiede solo 25 bytes!

```

                                :>#define _P1_B6 0x86
                                :>#define _P1_B7 0x87
                                :>bit unsigned char out_bit @ _P1_B6;
                                :>bit unsigned char in_bit @ _P1_B7;
                                :>
                                ...
                                : .segment __bang
                                : _bang: ; (leaf function) unsigned char
bang(unsigned char)
                                : ; parameter 'ob' in 'R7' assigned to 'R5'
co:8007: ad 07                   : mov R5,AR7
                                : ; variable 'ib' assigned to register 'R1'
                                :>
                                :>// Simple 8-Bit-Banger
co:8009: 7b 08                   :>unsigned char bang(unsigned char ob){
                                : mov R3,#8
                                : ??:
                                :> int i;
                                :> unsigned char ib;
                                :> for(i=0;i<8;i++){
co:800b: ed                       : mov A,R5
co:800c: 33                       : rlc A
co:800d: 92 86 : mov 134,C        :> out_bit=(ob&128);
                                :> ib<<=1;
co:800f: e9                       : mov A,R1
co:8010: 29                       : add A,R1
co:8011: f9                       : mov R1,A
                                :> if(in_bit) ib|=1;
co:8012: 30 87 03                 : jnb 135,?7
                                :
co:8015: 43 01 01                 : orl AR1,#1
                                : ??:
                                :> ob<<=1;

```

```
co:8018: ed          : mov A,R5
co:8019: 2d          : add A,R5
co:801a: fd          : mov R5,A
                   :> }
co:801b: db ee      : djnz R3,?3
                   :> return ib;
co:801d: af 01      : mov R7,AR1
co:801f: 22          : ret
                   : ; end of function bang
                   : ; used: R-1-3-5-7 BR----- ACC PSW
```

Come già detto, l'implementazione completa dell'ottimizzatore è ancora in corso, ci sono ancora margini per migliorarlo, le prossime versioni faranno sicuramente un lavoro ancora migliore.

APPENDICE A: INDICE ANALITICO

SIMBOLI

#ASM 20
#ENDASM 20
#PRAGMA 23
.END 31
.ENDMACRO 33
.ERROR 31
.EXPORT 32
.FILE 32
.HIDE 34
.IBYTES 31
.IF 33
.IFDEF 33
.IMPORT 31
.INCLUDE 31
.LINE 32
.MACRO 33
.SEGMENT 29
.SHOW 34
@ 21
__DATE__ 23
__FILE__ 23
__LINE__ 23
__STDC__ 23
__TIME__ 23
__UC__ 23
_I8051 23
_STARTUP_FIRST(VOID) 22

A

A51FLAGS 24
ADUC8XX 2
ANSI 12, 21
ASSEMBLER 19, 20
AVVIO RAPIDO 8

B

BIN_SAFE.H 38
BINARIO 11
BIT 13
BUG 11

C

C51FLAGS 13, 24
CHAR 12
CODE 13
CONTROLLO DI INTEGRITA' 22
CTYPE.H 37

D

DATI FONDAMENTALI 11
DEMO 4
DESCRIZIONE DEI MAKEFILES 25
DIRETTIVE ASSEMBLER 29
DISINSTALLAZIONE 5
DOCUMENTAZIONE 6

F

FAR 14
FLASHMON 10
FLOAT 12
FORMATTATORE 17
FUNZIONI INDIRETTE 20

G

GRAFI DI CHIAMATA 17
grifo® 4

I

I/O SERIALE 25
IEEE 12
INDIRIZZI ASSOLUTI 21
INEAR 13
INSTALLAZIONE 5
INT 12
INTEGRITA' 22
INTERRUPTS 16
INTRODUZIONE 1
IRQ52.H 39

K

KAR.H 39
KEY51.EXE 6

L

L51FLAGS 10, 25
LARGE 13
LIBRERIE STANDARD 36
LIMITI 4
LINKER 11
LONG 12

M

MACRO 35
MAKE 25
MAKE 9
MAKEFILE 25
MAKEFILE 9
MATH.H 38
MNEMONICI 28
MODEL 25
MODELLI DI MEMORIA 13, 25
MODIFICATORI 14
MSC1210 2

N

NEAR 13

O

OPZIONI
DEL COMPILATORE 23
OS535 8
MODELLO DELLA MEMORIA 9

P

PFLAGS 25
POINTER 12
BIT 12
CODE 12
INEAR 12
NEAR 12
XDATA 12
PREDEFINITI 23
PREFAZIONE 2
PRINTF() 17, 25
PROCESSORI 2
PROMOZIONE DEGLI INTERI 21
PUNTATORI 12

R

REGISTRAZIONE 6

REGOLE

CREARE LE PROPRIE REGOLE 27

ESPLICITE 26

IMPLICITE 26

S

SFR 21

SHORT 12

SIGNED 12

SIMBOLI GENERATI 35

SIMBOLI PREDEFINITI 23

SIOTYPE 25

SLD51 10

SMALL 13

STARTUP() 22

STDARG.H 38

STDIO.H 36

STRING.H 37

T

TIPI 12

U

UC51.EXE 11

UC51.KEY 6

UMSHELL 9

UNSIGNED 12

V

VARIABILI 28

VIRGOLA MOBILE 12

X

XDATA 13