Strumenti di analisi di programmi ed esempi del loro uso

Esercitazione 08 di Sistemi dedicati

Docente: Giuseppe Scollo

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Magistrale in Informatica, AA 2018-19

Indice

argomenti dell'esercitazione

in questa esercitazione si trattano:

- cross-compilatori e strumenti di utilità GNU per l'analisi di programmi di basso livello
- esempi di uso di tali strumenti per diversi processori e sistemi di sviluppo di programmi:
  - per emulatore ARM VisUAL v. 1.27 con cross-compilatore GCC 3.2, GNU Binutils 2.13, e script ad-hoc
  - esecuzione con debugger su processore ARM Cortex-A9 (DE1-SoC HPS)
  - esecuzione con debugger su processore Nios II (DE1-SoC FPGA)
  - per processore ARM Cortex-A9 con cross-compilatore GCC 4.8.1 e GNU Binutils 2.23
  - per processore Nios II con cross-compilatore GCC 5.3.0 e GNU Binutils 2.25
- esperienza di laboratorio: riproduzione dell'esecuzione degli esempi e analisi del problema segnalato alla fine della lezione precedente

  sorgenti C e script per l'esecuzione dell'esperienza sono disponibili nell'archivio classroom.tgz, reperibile nella cartella crosstools/e08 dell'area dedicata di laboratorio

cross-compilatori e binutils GNU

lo sviluppo di programmi per sistemi dedicati è spesso condotto in macchine basate su un processore diverso dal *target*, cioè quello designato per la loro esecuzione

- un *cross-compilatore* è un compilatore che genera codice assemblativo e programmi eseguibili per un'architettura diversa da quella su cui viene eseguito
- proprio come gli usuali compilatori, i cross-compilatori sono spesso accompagnati da una collezione di programmi di utilità per l'analisi di programmi di basso livello

i cross-compilatori e i programmi di utilità qui considerati sono software libero GNU, che tipicamente hanno i nomi <target>-gcc per il compilatore, dove <target> indica l'architettura target, e <target>-<util> per l'utilità <util>, quale per esempio

- size: lista delle dimensioni delle sezioni e totale di un modulo oggetto o di un eseguibile
- objdump: contenuti di un modulo oggetto o di un eseguibile
- nm: lista dei simboli di un modulo oggetto o di un eseguibile

esempi per emulatore ARM VisUAL

un primo esempio mostra l'uso dell'emulatore ARM VisUAL v. 1.27 per la simulazione dell'esecuzione di un programma per il calcolo del GCD con l'algoritmo di Euclide (Listing 7.11 in Schaumont, con `main` modificato per il diverso sistema di simulazione)

un secondo esempio, illustrato nella prossima figura, presenta una realizzazione software del calcolo del delay Collatz, simile in funzionalità alle realizzazioni hardware prodotte nelle precedenti esperienze di laboratorio

- l'emulazione qui rivela un problema lessicale con il codice (pre-UAL) prodotto dal cross-compilatore

il programma visto alla fine della lezione precedente (Listing 7.4 in Schaumont), con l'aggiunta di una inizializzazione del contenuto dell'array, forma il terzo esempio

- il genere di problemi prodotto dalla compilazione senza ottimizzazione è ben più serio in questo caso

l'emulatore accetta un programma sorgente assembly conforme a un sottoinsieme dell'insieme di istruzioni definito dalla sintassi UAL (*Unified Assembly Language*)

in questi esempi si genera il programma assembly dal sorgente C mediante il cross-compilatore `arm-linux-gcc`, per ISA ARMv4, realizzata nel processore StrongARM

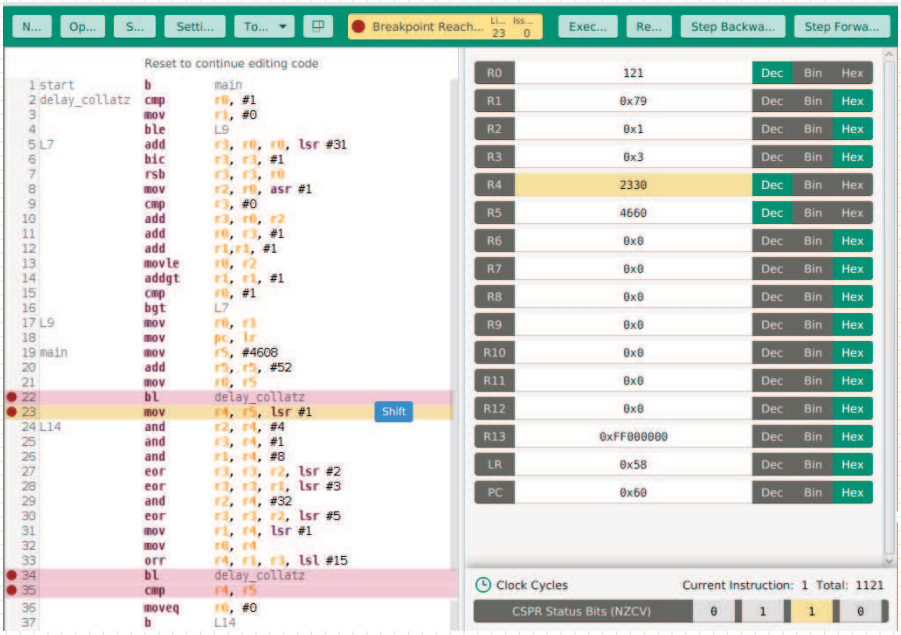
- l'installazione del pacchetto Debian fornisce sia il cross-compilatore sia i programmi di utilità

tuttavia, il sorgente assembly così prodotto non soddisfa le restrizioni imposte dall'emulatore, e.g contiene direttive di assemblatore e altri dettagli sintattici non accettati da VisUAL

- l'ostacolo si supera in buona parte mediante l'esecuzione di uno script ad-hoc, che modifica una copia del suddetto sorgente assembly producendone una versione *quasi* conforme alle restrizioni VisUAL

la cartella `arm_visual` nell'archivio fornito contiene i sorgenti C dei tre esempi e lo script `c2visual` per la cross-compilazione e modifica dell'assembly prodotto, come indicato sopra (lo script `c02visual` differisce dal precedente solo per la compilazione senza ottimizzazione, utile all'analisi del problema segnalato nella lezione precedente

emulazione VisUAL di un esempio



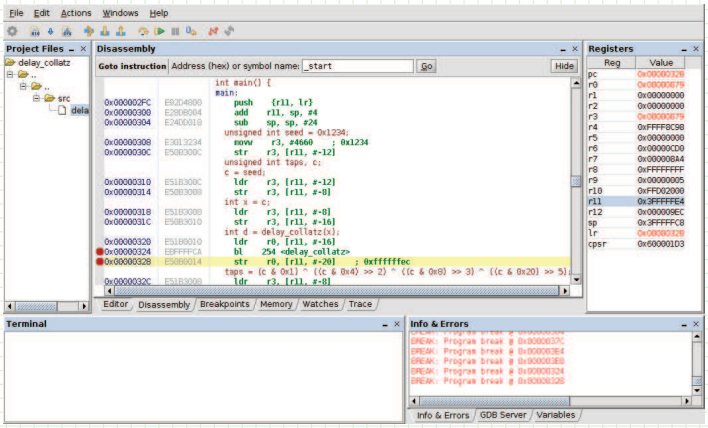
un'istantanea dell'emulazione VisUAL del programma assembly ARM `delay_collatz.s`

esecuzione con debugger per processore ARM Cortex-A9

la figura mostra un'istantanea dell'esecuzione del programma delay_collatz.c sul processore ARM della DE1-SoC

cross-compilazione, caricamento ed esecuzione hanno luogo mediante il programma altera-monitor-program, sotto controllo del suo debugger GNU GDB

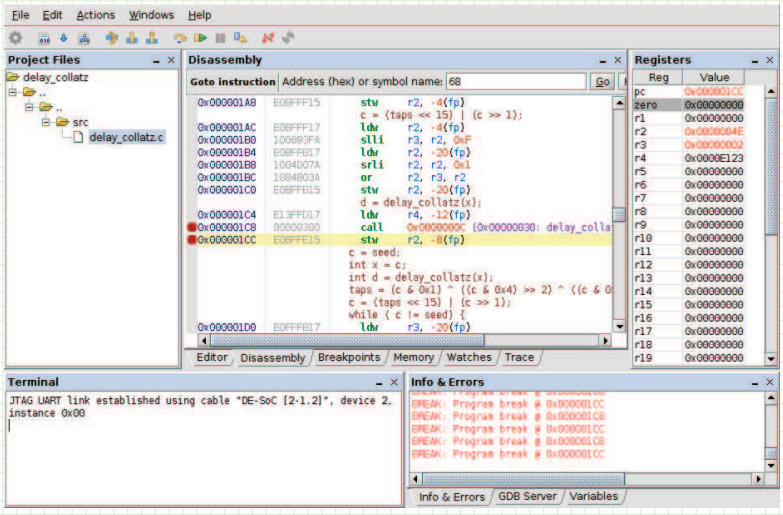
- il sistema permette il caricamento di un sorgente C o assembly e la cross-compilazione attraverso lo stesso programma monitor



istantanea di esecuzione di programma ARM controllata dal debugger GDB del Monitor Program

esecuzione con debugger per processore Nios II

lo stesso sorgente C dell'esempio precedente può essere cross-compilato ed eseguito da softcore Nios II sulla FPGA nel sistema DE1-SoC



istantanea di esecuzione di programma Nios II controllata dal debugger GDB del Monitor Program

l'esecuzione dell'esempio delay_collatz sul processore ARM con il monitor program genera nella cartella del progetto due file interessanti per l'analisi di basso livello:

- delay_collatz.axf: l'eseguibile in formato ELF
- delay_collatz.axf.objdump: il suo disassembly prodotto dall'utilità objdump

per condurre ulteriori analisi di basso livello è utile sapere dove sono collocati i programmi di utilità associati al cross-compilatore gcc utilizzato dal monitor program

> nella directory di installazione del software Quartus Prime, il cross-compilatore e le binutils sono reperibili nella cartella
> University_Program/Monitor_Program/arm_tools/baremetal/bin

N.B. in verità, il monitor program usa versioni specializzate di alcuni di questi strumenti, che hanno un'altra collocazione; tuttavia per l'uso generale, indipendente dall'esecuzione nel contesto del monitor program, è da utilizzare la collezione di strumenti indicata sopra

l'elaborazione dei sorgenti C con il monitor program per l'esecuzione sul processore Nios II genera l'eseguibile .elf nella cartella del progetto, ma non il suo disassembly

> nella directory di installazione del software Quartus Prime, il cross-compilatore e le binutils sono reperibili nella cartella
> nios2eds/bin/gnu/H-x86_64-pc-linux-gnu/bin

ecco un semplice esempio di uso dell'utilità size: confronto della dimensione degli eseguibili generati dai tre cross-compilatori qui considerati, per i sorgenti C di due degli esempi proposti e con le stesse opzioni di compilazione: in particolare, livello di ottimizzazione O2

> gli script usati a tal fine sono disponibili nella cartella sizes dell'archivio fornito

la tabella che segue presenta il risultato di questo esercizio

C source	cross-compiler	text	data	bss	total
gcd	arm-linux	820	260	4	1084
	arm-altera-eabi	492	16	28	536
	nios2-elf	1056	1068	0	2124
delay_collatz	arm-linux	864	260	4	1128
	arm-altera-eabi	628	16	28	672
	nios2-elf	1196	1068	0	2264

esperienza di laboratorio

riprodurre l'esecuzione degli esempi forniti per questa esercitazione, documentando eventuali difficoltà incontrate e le soluzioni trovate per superarle

in particolare, l'emulazione VisUAL dell'esempio `accumulate` richiede ulteriore editing del codice prodotto dallo script di compilazione + editing, sia con che senza ottimizzazione, per risolvere i seguenti problemi:

- con ottimizzazione (script `c2visual`): sembra che VisUAL ammetta solo *una* occorrenza della direttiva `DCD`; inoltre usa l'istruzione `ADR` (sconosciuta nell'ISA ARMv4) per il caricamento di un indirizzo in un registro, mentre nel codice prodotto dallo script si usa `LDR` da una locazione che detiene l'indirizzo in questione
- senza ottimizzazione (script `c02visual`): in aggiunta ai suddetti problemi, le istruzioni di trasferimento multiplo prodotte dallo script violano alcune restrizioni imposte da VisUAL coerentemente con simili prescrizioni introdotte in versioni dell'ISA ARM successive alla v4

in merito all'ultimo problema indicato, si propone di:

- risolverlo, ai fini dell'emulazione, usando l'istruzione `LDMFD sp! ...` invece di `LDMEA fp ...` per il ripristino dei registri e il rientro, con conseguenti modifiche alla gestione dell'area di attivazione e dei registri `fp` e `sp`
- spiegare perché la gestione della pila e dell'area di attivazione effettuata dal codice ARMv4 prodotto dal cross-assemblatore sia corretta, al contrario di figura 7.9, che vorrebbe illustrarla

riferimenti

letture raccomandate:

Schaumont, Ch. 7, Sect. 7.4

materiali per consultazione:

Schaumont, Ch. 7, Sect. 7.5, 7.6

The ARM Instruction Set, tutorial, ARM University Program, V1.0

VisUAL - A highly visual ARM emulator

tutorial di fonte Intel® FPGA University Program (Novembre 2016):

Introduction to the ARM® Processor Using Intel FPGA Toolchain - For Quartus Prime 16.1

Introduction to the Intel Nios II Soft Processor - For Quartus Prime 16.1

Intel FPGA Monitor Program Tutorial for ARM - For Quartus Prime 16.1

Intel FPGA Monitor Program Tutorial for Nios II - For Quartus Prime 16.1

materiali utili per l'esperienza di laboratorio proposta:

ARM Architecture Reference Manual, ARMv7-A and ARMv7-R Edition, ARM DDI 0406C.c (2014)

GCC, the GNU Compiler Collection

GNU Binary Utilities