

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 2017-18

Indice

1. Strumenti di analisi di programmi ed esempi del loro uso
2. argomenti dell'esercitazione
3. cross-compiler e binutils GNU
4. esempio per emulatore ARM VISUAL (1)
5. esempio per emulatore ARM VISUAL (2)
6. esecuzione con debugger per processore ARM Cortex-A9
7. esecuzione con debugger per processore Nios II
8. analisi di basso livello di eseguibili ARM
9. analisi di basso livello di eseguibili Nios II
10. esperienza di laboratorio
11. riferimenti

in questa esercitazione si trattano:

- cross-compiler 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 simulatore ARM VisUAL v. 1.27 con cross-compiler 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-compiler GCC 4.8.1 e GNU Binutils 2.23
 - per processore Nios II con cross-compiler GCC 5.3.0 e GNU Binutils 2.25
- esperienza di laboratorio: riproduzione dell'esecuzione degli esempi e produzione di documentazione esplicativa

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

i cross-compiler 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

esempio per emulatore ARM VisUAL (1)

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)

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

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

il cross-compilatore e i programmi di utilità sono reperibili da
rijndael.ece.vt.edu/gezel2repo/pool/main/a/arm-linux-gcc/

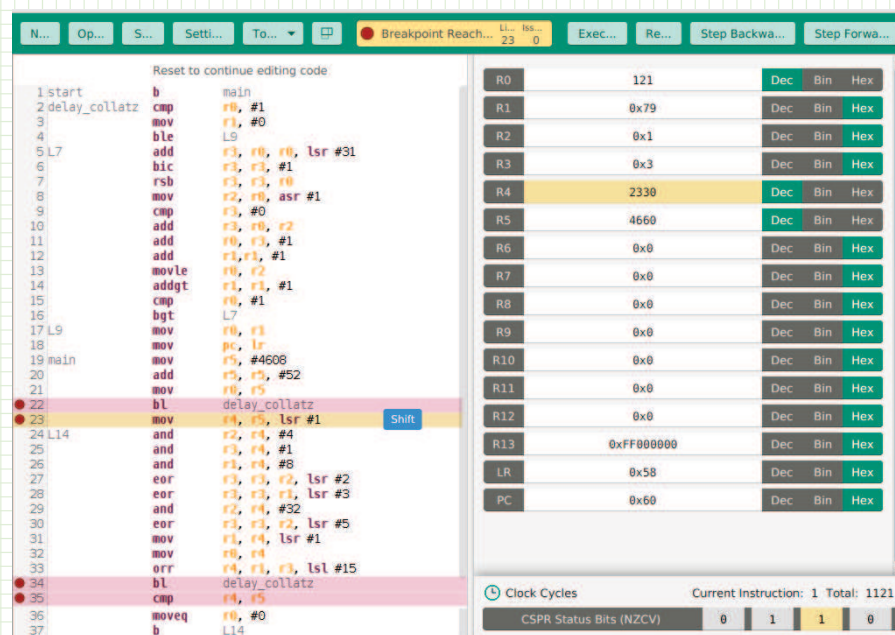
tuttavia, il sorgente assembly così prodotto non soddisfa le restrizioni imposte dall'emulatore, e.g. contiene direttive di assemblatore e altri dettagli sintattici non accettati da VisUAL

l'ostacolo si supera mediante uno script ad-hoc, che modifica una copia del suddetto sorgente assembly producendone una versione conforme alla sintassi VisUAL

l'archivio arm_visual_examples.tgz, reperibile nella cartella crosstools dell'area dedicata di laboratorio, fornisce i sorgenti C e gli script richiesti per la cross-compilazione e l'emulazione VisUAL dell'esempio GCD e di un secondo esempio, illustrato dalla prossima figura, che presenta una realizzazione software del calcolo del delay Collatz, simile in funzionalità alle realizzazioni hardware considerate nelle precedenti esperienze di laboratorio

per riprodurre l'esecuzione, dopo la decompressione dell'archivio con `tar xzpvf`, nella cartella `arm_visual_examples` eseguire rispettivamente `./c2visual gcd`, e `./c2visual delay_collatz`, quindi lanciare VisUAL e aprire i rispettivi file generati `gcd/gcd.s`, e `delay_collatz/delay_collatz.s` (attenzione: con estensione `.s` minuscola!)

esempio per emulatore ARM VisUAL (2)



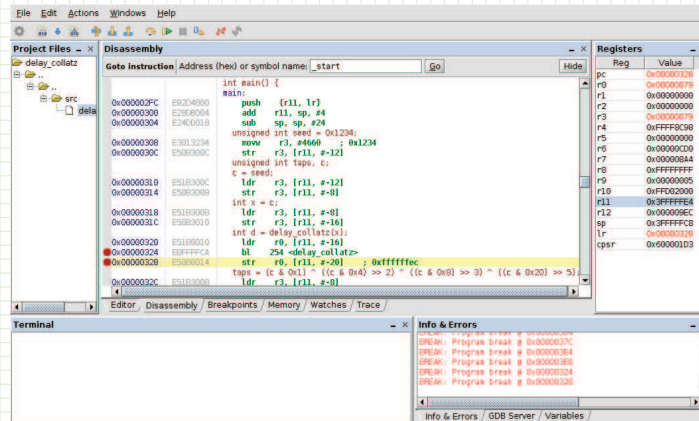
un'istantanea dell'emulazione VisUAL del programma assembly ARM `delay_collatz.s`

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

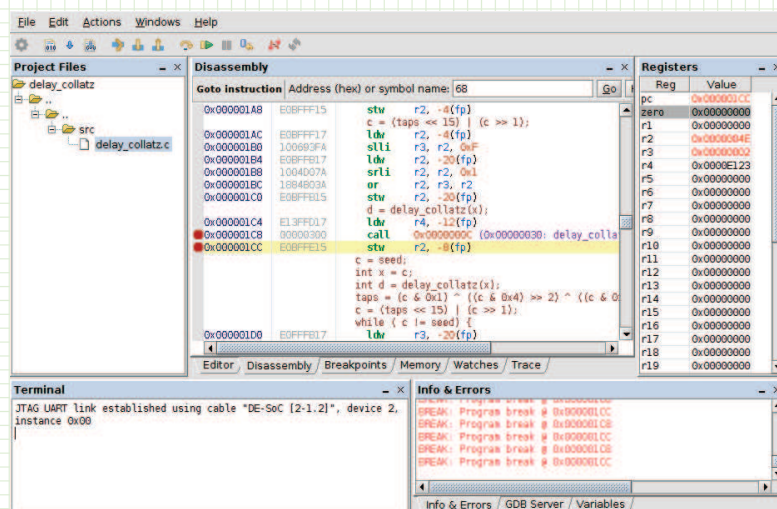
il sistema permette il caricamento di un sorgente C o assembly e la cross-compilazione attraverso lo stesso programma monitor



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 entrambi i sorgenti C proposti e con le stesse opzioni di compilazione: in particolare, livello di ottimizzazione `O2`

i due sorgenti e gli script usati a tal fine sono disponibili nell'archivio `cross-compiled_sizes.tgz`, reperibile nella cartella `crosstools` dell'area dedicata di laboratorio

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, esaminare i file prodotti dalla loro esecuzione, individuare aspetti poco noti dei loro contenuti, reperire in rete informazioni su di essi e produrre della documentazione esplicativa in forma di lista di domande e risposte

nella risposta va indicato un riferimento alla fonte; se la risposta è tratta da una fonte direttamente accessibile in rete, inserire il link specifico; se questo rinvia solo a una risposta alla domanda, il link è sufficiente come risposta

ecco tre esempi di domanda e risposta, uno per ciascuno dei tre tipi descritti sopra:

Q: Cosa indicano gli acronimi VMA e LMA nell'output di objdump?

A: Rispettivamente *virtual memory address* e *load memory address*; il primo è l'indirizzo di una sezione del programma durante l'esecuzione, il secondo è il suo indirizzo al primo caricamento del programma in memoria; i due indirizzi possono differire se il programma è caricato in una memoria diversa da quella che lo contiene a tempo di esecuzione, per esempio caricato su una memoria Flash e copiato su RAM quando va in esecuzione. (Schaumont, 2012, pp. 216-217)

Q: Cosa fa l'istruzione ARM `rsb`?

A: Sta per *reverse subtract*: sottrae il primo operando sorgente dal secondo, ovvero minuendo e sottraendo sono in ordine inverso rispetto alla `sub`; si noti che nelle istruzioni aritmetiche ARM il secondo operando sorgente può essere immediato, mentre il primo e la destinazione devono essere registri, dunque questa istruzione permette la sottrazione da una costante. (www.heyrick.co.uk/armwiki/RSB)

Q: Qual è l'origine del nome `bss` della sezione di un programma eseguibile che contiene le variabili globali?

A: it.wikipedia.org/wiki/.bss

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:

GCC, the GNU Compiler Collection

GNU Binary Utilities