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 2016-17

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 processore NIOS II con cross-compilatore e utilità GCC 4.7.3 e 5.2.0
  - per simulatore SimIt-ARM v. 2.1-sfu con cross-compilatore e utilità GCC 3.2
  - per processore ARM Cortex-A9 con cross-compilatore e utilità GCC 5.2.0
  - esecuzione con debugger per processore NIOS II su FPGA
- esperienza di laboratorio: riproduzione dell'esecuzione degli esempi e produzione di documentazione esplicativa

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

esempio per processore NIOS II

l'uso del cross-compilatore nios2-elf-gcc e di un paio di programmi di utilità, illustrato in sez. 7.4 di (Schaumont 2012) per l'esempio nel Listing 7.7, è riproducibile con gli script nell'archivio nios2binutils_example.tgz, reperibile nella cartella crosstools dell'area dedicata di laboratorio

> gli script sono disponibili in due versioni, ciascuna relativa alla versione (13.1 o 16.0) della suite nios2eds che contiene il cross-compilatore e relativi programmi di utilità, rispettivamente delle versioni 4.7.3 e 5.2.0 di GCC

per ciascuna versione sono disponibili due script:

- nios2gccToolchain.sh per la produzione di sorgente assemblativo, oggetto ed eseguibile
- ext_build/nios2gcc_extToolchain.sh per l'analisi del file oggetto e dell'eseguibile con le utilità size e objdump e la generazione della mappa della memoria del linker

esempio per simulatore SimIt-ARM

l'esempio mostra l'uso del simulatore SimIt-ARM v. 2.1 sia per l'analisi dell'eseguibile che per la simulazione, da linea di comando, di un programma per il calcolo del GCD con l'algoritmo di Euclide (Listing 7.11 in (Schaumont, 2012))

> si usa la versione 2.1 del simulatore invece della più recente 3.0 perché la prima permette la simulazione al livello di ciclo, mentre la seconda solo al livello di istruzione

la compilazione usa il cross-compilatore arm-linux-gcc, per ISA ARMv5, realizzata fra altri dal processore StrongARM

> sia il cross-compilatore con relativi programmi di utilità sia il simulatore sono reperibili nel repository rijndael.ece.vt.edu/gezel2repo

l'archivio arm_simitarm_example.tgz, reperibile nella cartella crosstools dell'area dedicata di laboratorio, fornisce due script per l'esercitazione:

- arm-linux-gccToolchain.sh per la produzione di sorgente assemblativo, oggetto ed eseguibile
- sim/SimIt-ARM.sh per l'analisi dell'esecuzione al livello di ciclo (simulatore sima) e per la simulazione con debugger ema, con input dei comandi nel file emadCommands.txt

esempio per processore ARM Cortex-A9

l'esempio mostra l'esecuzione su ARM Cortex-A9 nel sistema Altera DE1-SoC di una variante del programma nel Listing 7.1, in cui il corpo della funzione main() è costituito da

```
printf("maxgcd(a,b)=%d\n", gcd(a,b));
return 0;
```

il caricamento e l'esecuzione hanno luogo mediante il programma altera-monitor-program, sotto controllo del debugger GDB in esso incorporato e mediante la connessione JTAG del programma alla DE1-SoC

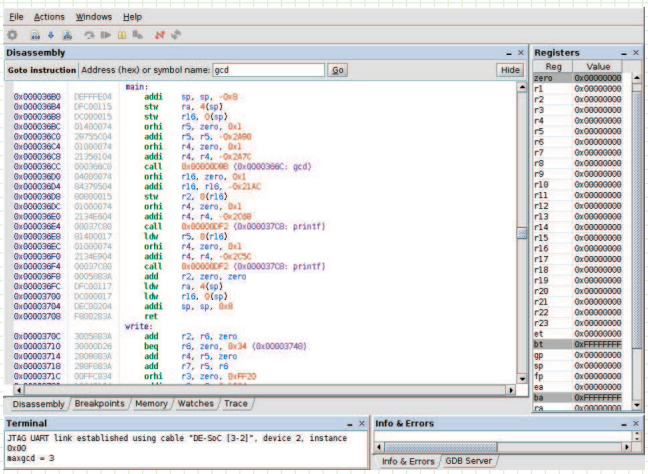
> il sistema permette il caricamento di un sorgente C o assembly e la compilazione attraverso lo stesso programma monitor, tuttavia in questo esempio si forniscono script analoghi a quelli degli esempi precedenti, per consentire l'analisi con programmi di utilità, e si carica l'eseguibile generato dagli script

l'archivio arm_cortex-a9_example.tgz reperibile nella cartella crosstools dell'area dedicata di laboratorio, contiene gli script:

- arm_altera_gccToolchain.sh per la produzione di sorgente assemblativo, oggetto ed eseguibile
- ext_build/arm_altera_gcc_extToolchain.sh per l'analisi del file oggetto e dell'eseguibile

esecuzione con debugger per processore NIOS II su FPGA

lo stesso sorgente C dell'esempio precedente può essere compilato ed eseguito da softcore NIOS II sulla FPGA nel sistema DE1-SoC, in questo esempio senza alcun altro script bensì per esplorare l'uso del debugger incorporato nel programma monitor



un'istantanea di esecuzione su board Altera DE1-SoC, controllata dal debugger dell'Altera Monitor Program

esperienza di laboratorio

eseguire gli script 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 (2012) Cap. 7, Sez. 7.4, 7.6.2

letture per ulteriori approfondimenti:

> Schaumont (2012) Cap. 7, Sez. 7.6.1, 7.6.3

materiali utili per l'esperienza di laboratorio proposta:

> *GCC, the GNU Compiler Collection*
>
> *GNU Binary Utilities*
>
> *Introduction to the Altera Nios II Soft Processor*, Altera University Program, May 2016
>
> *Introduction to the ARM® Processor Using Altera Toolchain*, Altera University Program, May 2016
>
> SimIt-ARM Users' Guide
>
> *Altera Monitor Program Tutorial for Nios II*, Altera University Program, May 2016
>
> *Altera Monitor Program Tutorial for ARM*, Altera University Program, May 2016