Progetto SO 2019/2020

November 25, 2019

BiKaya OS

- BiKaya: Evoluzione di Kaya O.S., a sua volta evoluzione di una lunga lista di S.O. proposti a scopo didattico (HOCA, TINA, ICARO, etc).
- BiKaya deve essere realizzato su architettura emulata
- Architettura basata su seilivelli di astrazione, sul modello del S.O. THE proposto da Dijkstra in un suo articolo del 1968 ...

6 Livelli di Astrazione

Livello 6: Shell interattiva

Livello 5: File System

Livello 4: Livello di Supporto

Livello 3: Kernel del S.O.

Livello 2: Gestione delle Code

Livello 1: Servizi offerti dalla ROM

Livello 0: Hardware

- Fontamenta note
- Fase 1

6 Livelli di Astrazione

Livello 6: Shell interattiva

Livello 5: File System

Livello 4: Livello di Supporto

Livello 3: Kernel del S.O.

Livello 2: Gestione delle Code

Livello 1: Servizi offerti dalla ROM

Livello 0: Hardware

- Fontamenta note
- Fase 2

Variazione

- Ogni anno la consegna varia da quello precedente
- Quest'anno dovrete sviluppare il vostro lavoro su due architetture anziche' su una sola
- La mole di lavoro sara' regolata adeguatamente
- I due emulatori sono μMPS2 e μARM

Cosa significa avere come target multiple architetture?

- Ci si aspetta che il vostro codice possa essere compilato per due macchine diverse mantenendo le stesse funzionalita'.
- La complessita' aggiunta consiste nel gestire le differenze sottostanti alla logica applicativa, astraendo il piu' lontano possibile da esse.
- Alcune parti del codice saranno necessariamente diverse per le due destinazioni; dovrebbero pero' essere poche e contenute.

Come?

- Le varianti della compilazione dovrebbero essere gestite dal build system (make, cmake, scons, ...)
- Come per il codice, le procedure sono simili ma sostanzialmente diverse: servono due compilatori (mipsel-linux-gnu e arm-none-eabi)
- Il vostro script di compilazione dovra' operare diversamente in base a un parametro che indica il target di compilazione

Esempio

```
#ifdef TARGET_UMPS
// Codice specifico a uMPS2
#elif defined(TARGET_UARM)
// Codice specifico a uARM
#endif
```

Usando il flag -D durante la compilazione e' possibile definire una macro on-the-fly:

```
$ gcc -DTARGET_UMPS ...
```

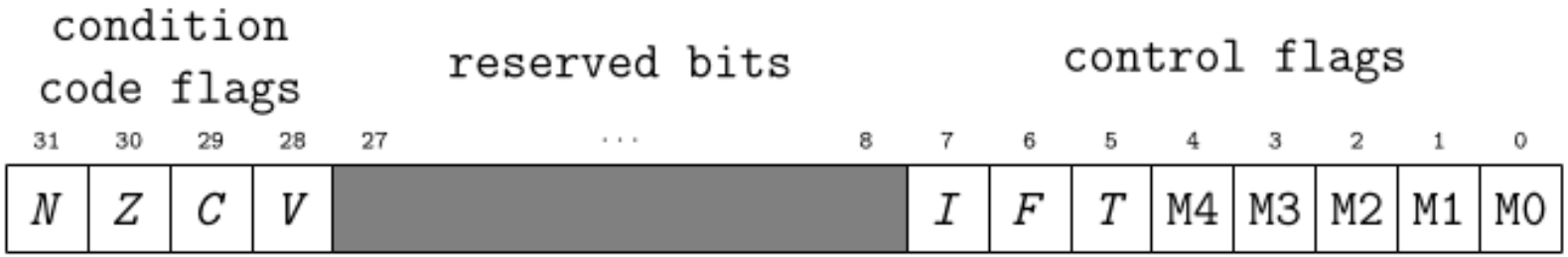
Differenze tra μMPS2 e μARM

In realta', molto poche:

1. Indirizzi di memoria dove trovare i registri dei dispositivi
2. Registri di sistema (in particolare STATUS e CAUSE)
3. Stato dei processi
4. I dispositivi funzionano allo stesso modo
5. La maggior parte delle altre differenze non sono rilevanti per il lavoro che farete

Molto probabilmente una volta gestite le differenze di base non ci saranno errori che si presentano solo su uno dei due emulatori.

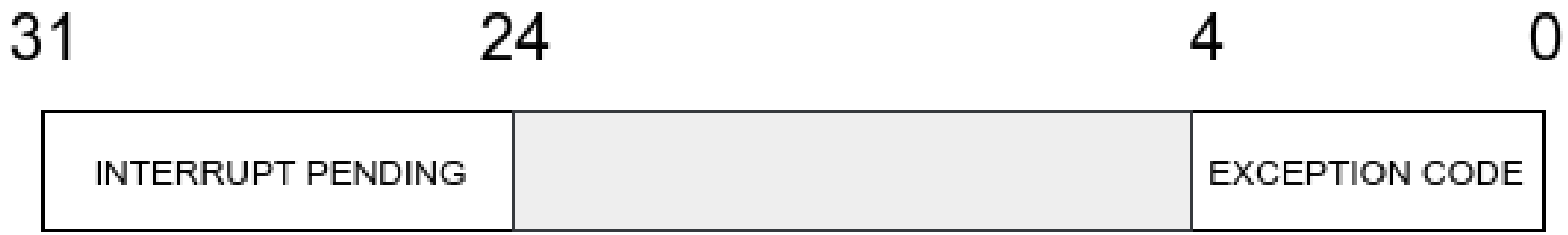
Status Register



μARM status register

μMPS2 status register

Cause Register



μARM cause register

μMPS2 cause register

Esempio

`https://github.com/Maldus512/umps_uarm_hello_world/tree/example`

```
$ git clone https://github.com/Maldus512/umps_uarm_hello_world.git
$ git checkout example
```

Build System

- Siete fortemente incoraggiati a usare il build system che preferite
- Make e' uno standard di fatto, SCons e' piu' moderno e semplice da utilizzare, CMake e' piu' potente e professionale...
- Riuscire a compilare correttamente il codice fa parte del progetto!

Consegna

- La primissima parte del progetto e' la Fase 0 (da non confondere con il livello 0 di Kaya); serve a prendere confidenza con gli emulatori.
- Dovrete costruire un programma che legga una stringa dal terminale 0 fino al carattere \n e che la scriva integralmente sulla stampante 0.
- Complessivamente si tratta di un esercizio che dovrebbe richiedere al massimo 100 righe di codice.

Installazione

- `https://github.com/mellotanica/uARM`
- `http://mellotanica.github.io/uARM/`
- `https://github.com/tjonjic/umps`
- `https://github.com/Zimm1/umps-apt-installer/blob/master/installUmps.sh`

Pacchetti richiesti: git libtool m4 qt5-qmake make automake autotools-dev autoconf qt5-base qt4-default libelf-dev libboost-dev libelf1 libsigc++-2.0-dev mipsel-linux-gnu arm-none-eabi

Compilazione per μMPS2

```
$ mipsel-linux-gnu-gcc -ansi -mips1 -mfp32 -I/usr/local/include/umps2 -c
???.c
$ mipsel-linux-gnu-ld -o kernel ???.o /usr/local/lib/umps2/crtso.o
/usr/local/lib/umps2/libumps.o -nostdlib -T
/usr/local/share/umps2/umpscore.ldscript
$ umps2-elf2umps -k kernel
```

Compilazione per μARM

```
$ arm-none-eabi-gcc -O0 -mcpu=arm7tdmi -I/usr/include/uarm -c ???.c
$ arm-none-eabi-ld -o kernel ???.o -G0 -nostdlib
-T/usr/include/uarm/elf32ltsarm.h.uarmcore.x
```