SCALETTA LEZIONE SERT 25.10.2022 (E05)

Perché ciclo vuoto, gcc ottimizza!

1 Modificare noop.c per far lampeggiare i LED
  1.1 In ARM-335x 25.4.1.25: il registro GPIO_CLEARDATAOUT
      all'offset 0x190 serve a deasserire una linea di GPIO
  1.2 Prima versione:

```
|int c, state = 0;                                    |
|for(;;) {                                            |
|    for (c=0; c<100000000; ++c)                      |
|        ;                                            |
|    gpio1[0x190/4] = 0xf << 21;   //config linea uscita (shift 21 perché opero sui bit dal 21 al 24)
|    gpio1[0x194/4] = state << 21; //accensione led   |
|    state = (state+1) & 0xf;                         |
|}                                                    |
```

Lo azzero ogni 15 secondi, evito overflow

      Funziona? NO!
   1.3 Controllare il listato assembly (in sert.lst)
   1.4 Aggiungere 'volatile' alla definizione di gpio1
   1.5.Aggiungere asm("") al ciclo interno
   1.6 Funziona? SI!
   1.7 ATTENZIONE! Il programma funziona solo per caso!
       1.7.1 Le variabili automatiche sono poste di norma sullo stack,
             che non e' stato inizializzato. In questo caso sono cosi'
             poche che il compilatore ha usato solo registri

GPIO1 è stato inizializzato da uBoot, ma non posso sempre "fidarmi" di lui.
Dal manuale del Chipset (No 8888) vedo registri e circuito che distribuiscono il clock.

2 Organizzazione del codice sorgente:
  2.1 Organizzazione degli header file
      2.1.1 Header file principale: beagleboneblack.h
      2.1.2 Sub-header file con definizioni per software: comm.h
      2.1.3 Sub-header file con definizioni per hardware: bbb-xxx.h
  2.2 Trasformazione di noop.c in main.c

```
|#include "beagleboneblack.h"
|volatile int *gpio1 = (int *) 0x4804c000;
|volatile int *cm_per = (int *) 0x44e00000;
|void init_gpio1(void)
|{
|        cm_per[0xac/4] = 0x40002;
|        gpio1[0x134/4] &= ~((1<<21)|(1<<22)|(1<<23)|(1<<24));
|}
|void loop_delay(unsigned long d)
|{
|        unsigned int c;
|        for (c=0; c<d; ++c)
|                compiler_barrier();
|}
|void _reset(void)
|{
|        int state = 1;
|        init_gpio1();
|        for(;;) {
|                loop_delay(100000000);
|                gpio1[0x190/4]  = 0xf << 21;
|                gpio1[0x194/4]  = state << 21;
|                state = (state+1) & 0xf;
|        }
|}
```

offset in byte /4 (perché uso vettore di interi)

Se settato a 0, è spento e porta al reset (dai docum.)

Per gcc nessuno li usa, gcc non sa che sono registri, devo antecedere asm volatile (" "), ovvero dico a gcc di non toccare!

Lo faccio anche con i vettori: volatile int *cm_per, così gcc non li tocca!

  2.3 Funziona? NO! Leggendo sert.lst scopriamo che _reset() potrebbe non
      essere all'indirizzo 0x80000000! (3 funzioni prima di reset)

3 Modifichiamo sert.lds in modo che '_reset' sia sempre posto all'inizio
  della RAM:

```
|[...]                                                |
|SECTIONS                                             |
|{                                                    |
|    . = mem_start;                                   |
|    .text : {                                        |
|        startup.o(.text) <- mettila prima di tutti   |
|        . = ALIGN(4);                                |
|        *(.text)                                     |
|        . = ALIGN(4);                                |
|    } > ram                                          |
|[...]                                                |
```

  3.1 Cambiamo il nome della funzione in main.c da _reset() a main()
  3.2 Definiamo un file assembler startup.S con la funzione _reset()

```
|    .text                                            |
```

-> per semplificare

```
|    .code 32 = arm 32 bit                                    |
|    .global _reset = serve al linker per trovarlo            |
|_reset:                                                      |
|    b       main   (b branch per ARMv7-A)                    |
+-------------------------------------------------------------+
```

3.3 Controllare la posizione di _reset() in sert.sym . Stack non inizializzato, non funge!
↳ non posso introdurre salti e funzioni senza considerarlo!

4 Inizializzazione dello stack
4.1 Modificare sert.lds in modo da definire uno stack dopo la sezione dati non inizializzati:

```
+-------------------------------------------------+
|        _bss_end = .;                            |
|    } > ram                                      |
|    .stack : {                                   |
|        . = ALIGN(4096);                         |
|        stack_end = .;                           |
|        . = . + 4096;                            |
|        stack_top = .;                           |
|    } > ram                                      |
|}                                                |
+-------------------------------------------------+
```

= lascio libero uno spazio per lo stack, il quale parte dall'alto (INDIRIZZO ALTO) verso il basso (INDIRIZZO BASSO)

4.2 Aggiungere in startup.S la inizializzazione dello stato della CPU:

```
+-------------------------------------------------+
|         .text                                   |
|         .code 32                                |
| #define         SYS_MODE        0x1f = user privilegiato
|         .global _reset                          |
| _reset:                                         |
|         cpsid           if,#SYS_MODE            |
|         ldr             sp,=stack_top = automatizzo le op. di registro alto e reg. basso.
|         b               main                    |
+-------------------------------------------------+
```

Memory Ordering : ordine istr ≠ ordine esecuz.
SOLUZIONE: mem. barriers.

5 Riscrivere loop_delay() con una barriera di memoria completa
5.1 Una barriera di memoria puo' avere diverse funzioni:
5.1.1 Impedire che il compilatore ottimizzi e riordini la sequenza delle istruzioni a livello di codice sorgente
5.1.2 Impedire che il processore riordini nella pipeline la sequenza di istruzioni macchina effettivamente eseguite
5.1.3 Impedire che il processore riordini la sequenza di accessi alla memoria
5.1.4 Impedire che il processore permetta l'esecuzione di una istruzione macchina prima del completamento di un accesso alla memoria precedente
5.1.5 Riferimento: "ARM Cortex-A Series Programmer's Guide", 10.2
5.2 Aggiungiamo al file bbb_cpu.h la definizione di macro che realizzano queste barriere:

```
+-------------------------------------------------------------------+
|#define compiler_barrier() \                                       |
|    __asm__ __volatile__ ("" ::: "memory")                         |
|#define data_memory_barrier() \                                    |
|    __asm__ __volatile__ ("dmb sy" ::: "memory")                   |
|#define data_sync_barrier() \                                      |
|    __asm__ __volatile__ ("dsb sy" ::: "memory")                   |
|#define instr_sync_barrier() \                                     |
|    __asm__ __volatile__ ("isb sy" ::: "memory")                   |
+-------------------------------------------------------------------+
```

5.3 Riscrivere loop_delay() in comm.h utilizzando data_sync_barrier():

```
+-------------------------------------------------------------------+
|static inline = Poniz. statica rispetto al file.                   |
|void loop_delay(unsigned long d)                                   |
|{                                                                  |
| while (d-- > 0)                                                   |
|     data_sync_barrier();                                          |
|}                                                                  |
+-------------------------------------------------------------------+
```

5.3.1 Rimuovere la definizione di loop_delay() in main.c
5.3.2 La nuova primitiva di sincronizzazione rende ogni iterazione del ciclo molto piu' lenta, quindi abbassiamo il numero di cicli eseguiti da 100000000 a 10000000

6 Creiamo un file 'init.c' per l'inizializzazione dell'ambiente di esecuzione
6.1 Definiamo una funzione _init() per eseguire le inizializzazioni e saltare poi a main
6.1.1 Modificare _reset in modo che salti a _init invece di main
6.1.2 Aggiungere a comm.h il prototipo di main()
6.1.3 Scrivere la funzione _init():

```
+-------------------------------------------------------------+
|void _init(void)                                             |
```

```
|{
|    fill_bss();
|    main();
|}
```

deve inizializzarlo IO a ø!

6.1.4 Scrivere la funzione fill_bss():

```
|static void fill_bss(void)
|{
|    extern u32 _bss_start, _bss_end;
|    u32 *p;
|    for (p = &_bss_start; p < &_bss_end; ++p)
|        *p = 0UL;
|}
```

6.1.5 Definire il tipo di dati u32 in beagleboneblack.h: (Nello ~~BBB~~ X questo tipo di dato!)

```
|typedef unsigned int u32;
```

7 Definiamo un metodo uniforme per accedere ai registri delle periferiche

7.1 Metodo inefficiente:

```
|volatile u32 * const gpio1  = (u32 *) 0x4804c000;
|#define GPIO_DATAOUT (0x13c/4)
|gpio1[GPIO_DATAOUT] = state;
```

7.1.1 E' inefficiente perche' ogni accesso ad un registro di periferica richiede (1) un accesso in RAM per leggere la variabile gpio1, e (2) un accesso alla memoria di I/O della periferica

7.1.2 D'altra parte e' comodo avere vettori u32 e volatile per far eseguire al compilatore i controlli di consistenza sui tipi di dati

7.2 La nostra soluzione:

7.2.1 Aggiungiamo in beagleboneblack.h:

```
|static volatile u32 *const _iomem = (u32 *) 0;
|#define iomemdef(N,V) enum { N = (V)/sizeof(u32) };
|#define iomem(N) _iomem[N]
|static inline void iomem_high(unsigned int reg, u32 mask)
|{
|    iomem(reg) |= mask;
|}
|static inline void iomem_low(unsigned int reg, u32 mask)
|{
|    iomem(reg) &= ~mask;
|}
|
```

7.2.2 Creiamo un nuovo file bbb_gpio.h contenente:

```
|#define GPIO1_BASE            0x4804c000
|iomemdef(GPIO1_OE,               GPIO1_BASE + 0x134);
|iomemdef(GPIO1_DATAOUT,          GPIO1_BASE + 0x13c);
|iomemdef(GPIO1_CLEARDATAOUT,     GPIO1_BASE + 0x190);
|iomemdef(GPIO1_SETDATAOUT,       GPIO1_BASE + 0x194);
|iomemdef(GPIO1_IRQSTATUS_CLR_0,  GPIO1_BASE + 0x3c);
|iomemdef(GPIO1_IRQSTATUS_CLR_1,  GPIO1_BASE + 0x40);
```

7.2.3 Per accedere al registro della periferica:
"iomem(GPIO1_DATAOUT)"

7.2.3.1 Il compilatore ottimizza cancellandola l'operazione di somma della base del vettore (0) e quindi viene effettuato solo un accesso alla memoria di I/O

7.3 Spostiamo l'inizializzazione del modulo GPIO1 in init.c:

```
|static void init_gpio1(void)
|{
|     u32 mask = (1 << 21) | (1 << 22) | (1 << 23) | (1 << 24);
|     iomem(CM_PER_GPIO1_CLKCTRL) = 0x40002;
|     iomem_low(GPIO1_OE, mask);
|     iomem_high(GPIO1_IRQSTATUS_CLR_0, mask);
|     iomem_high(GPIO1_IRQSTATUS_CLR_1, mask);
|}
|void _init(void)
|{
|    init_gpio1();
|    fill_bss();
|    main();
|}
```

```
    +-------------------------------------------------------------+
  7.4 Definire un nuovo header file bbb_cm.h per il modulo CM_PER:
    +-------------------------------------------------------------+
    |#define CM_PER                  0x44e00000                  |
    |iomemdef(CM_PER_GPIO1_CLKCTRL,  CM_PER+0xac);                |
    +-------------------------------------------------------------+
  7.5 Modifichiamo main.c
    +-------------------------------------------------------------+
    |void main(void)                                              |
    |{                                                            |
    |    int state = 1;                                           |
    |    for(;;) {                                                |
    |        loop_delay(10000000);                                |
    |        iomem(GPIO1_CLEARDATAOUT) =  0xf << 21;              |
    |        iomem(GPIO1_SETDATAOUT) = state << 21;               |
    |        state = (state+1) & 0xf;                             |
    |    }                                                        |
    |}                                                            |
    +-------------------------------------------------------------+

==========

/*
vim: tabstop=3 softtabstop=4 expandtab list colorcolumn=74
*/
```