

In ARM ho 7 tipi di eccezione, spesso gestite diversamente da INTEL.

- 1 Aggiungere a `_init()` l'inizializzazione della tabella dei vettori di interruzione
 - 1.1 Riprendere slides lez-E03b per **modalita' gestione delle eccezioni**
 - 1.2 Consultazione del manuale "ARM Cortex-A8 TRM" per determinare la posizione della tabella dei vettori interruzione (3.2.25, 3.2.68)
 - 1.3 Scrittura della funzione `get_vectors_address()` in **bbb_vectors.h**:

In sintesi, questo codice legge il valore del registro di controllo (c1) del coprocessore di sistema ARM (p15) e lo memorizza nella variabile `v`, che usa un registro.

```
static inline u32 *get_vectors_address(void)
{
    u32 v;
    __asm__ __volatile__ ("mrc p15, 0, %0, c1, c0, 0\n": "=r"(v)::);
    if (v & (1 << 13))
        return (u32 *) 0xffff0000;
    __asm__ __volatile__ ("mrc p15, 0, %0, c12, c0, 0\n": "=r"(v)::);
    return (u32 *) v;
}
```

metti in %0 il registro che andrà in 'v'

- 1.4 Scrittura della funzione `init_vectors()` in `init.c`

```
static void init_vectors(void)
{
    extern void _reset(void);
    volatile u32 *vectors = get_vectors_address();
    #define LDR_PC_PC 0xe59ff018
    vectors[0] = LDR_PC_PC; /* Reset / Reserved */
    vectors[1] = LDR_PC_PC; /* Undefined instruction */
    vectors[2] = LDR_PC_PC; /* Software interrupt */
    vectors[3] = LDR_PC_PC; /* Prefetch abort */
    vectors[4] = LDR_PC_PC; /* Data abort */
    vectors[5] = LDR_PC_PC; /* Hypervisor trap */
    vectors[6] = LDR_PC_PC; /* Interrupt request (IRQ) */
    vectors[7] = LDR_PC_PC; /* Fast interrupt request (FIQ) */
    vectors[8] = (u32) _reset; /* Reset / Reserved */
    vectors[9] = (u32) panic0; /* Undefined instruction */
    vectors[10] = (u32) panic0; /* Software interrupt */
    vectors[11] = (u32) panic0; /* Prefetch abort */
    vectors[12] = (u32) panic1; /* Data abort */
    vectors[13] = (u32) panic0; /* Hypervisor trap */
    vectors[14] = (u32) panic0; /* Interrupt request (IRQ) */
    vectors[15] = (u32) panic0; /* Fast interrupt request (FIQ) */
    #undef LDR_PC_PC
}
```

Problema: ho 32 bit = 4 byte per codificare il punto a cui saltare. In questi 4 byte non riesco ad inserire l'istruzione di salto + codice salto.

Se accedo a `vectors[0]` vado a `vectors[2]`, poi a [4] fino a 8, poichè vado di due istruzioni alla volta. Faccio così per problemi di spazio.

Per ARM posso usare anche i `vectors` da 8 a 15. Vediamo molte funzione "panic", non li passo con parametro perchè non ho spazio.

- 1.5 Aggiungere invocazione di `init_vectors()` in `_init()`
 - 1.6 Prova del funzionamento: aggiungere `asm("swi 0")` in `main()`
 - 1.7 Sostituire `panic0()` con `panic2()` in `vector[10]`
- 2 Consultare i manuali per informazioni su come si programma la porta seriale "UART"
 - 2.1 Il manuale "AM335x TRM" dedica piu' di 100 pagine alla programmazione delle porte seriali. Tre possibilità:
 - 2.1.1 Programmazione tramite DMA: nel trasferimento dei dati non è coinvolta la CPU Direct Memory Access, difficile

obiettivo: trovare il meccanismo per far apparire sul terminale i messaggi sulla porta seriale

2.1.2 Programmazione tramite interruzioni: la CPU trasferisce i dati, le interruzioni segnalano la presenza di dati in ricezione o la disponibilit  di spazio nel buffer di trasmissione

2.1.3 Programmazione tramite CPU polling: la CPU trasferisce i dati e controlla attivamente lo stato dei buffer di ricezione/trasmissione

2.2 Scegliere la strada pi  semplice: programmare la porta seriale tramite CPU polling. Determinare posizione e funzionamento dei registri di I/O della porta seriale, in particolare il registro di dati e quello che informa sullo stato del buffer di trasmissione

2.2.1 Registro THR (Transmit Hold Register)

Scrittura: dato da trasmettere su porta seriale (bit 0-7 del registro)

2.2.2 Registro LSR (Line Status Register)

Bit 5: TXFIFOE (Transmit Hold Register Empty)

mi dice QUANDO posso scrivere, lo metto nel quinto bit: 0 non posso, 1 posso.

2.3 Scrivere bbb_uart.h aggiungendo le definizioni occorrenti

```
+-----+
| #define UART0_BASE          0x44e09000
| iomemdef(UART0_THR,         UART0_BASE + 0);
| iomemdef(UART0_LSR,         UART0_BASE + 0x14);
| #define LSR_TXFIFOE         (1u<<5)
|-----+
```

2.3.1 Aggiungere include bbb_uart.h in beagleboneblack.h

2.4 Scrivere file uart0.c

2.4.1 Funzione putc() per scrivere un singolo carattere

```
+-----+
| int putc(int ch)
| {
|     while (!(iomem(UART0_LSR) & LSR_TXFIFOE))
|         /* do nothing */ ;
|     iomem(UART0_THR) = ch;
|     return 1;
| }
|-----+
```

2.4.2 Aggiungere extern per putc() in comm.h:

extern void putc(int);

2.4.3 Provare la funzione putc() modificando main()

2.4.4 Funzione puts() per scrivere una stringa di caratteri

```
+-----+
| int puts(const char *st)
| {
|     int v = 0;
|     while (*st) {
|         v += putc(*st);
|         if (*st++ == '\n')
|             v += putc('\r');
|     }
|     return v;
| }
|-----+
```

Se `""st++`, cio  il carattere successivo a quello in esame,   `"\n"`, allora al carattere attuale metto `"\r"`, in quanto la porta seriale usa un vecchio protocollo che usa `"\r\n"`.

Senza `\r` stampa:

```
SERT
SERT
1
cio  non ritorna a colonna 0.
```

2.4.5 Aggiungere extern per puts() in comm.h

2.5 Rinominare main() in endless_led_blinking()

2.6 Scrivere nuova main() che stampa su seriale ed invoca endless_led_blinking()

3 Scrivere la funzione putnl() per inserire 'new line':

```
+-----+
| int putnl(void)
| {
|     putc('\n');
|     putc('\r');
|     return 2;
| }
+-----+
```

4 Scrivere la funzione puth() in uart0.c per stampare un valore senza segno in **esadecimale**

```
+-----+
| int puth(unsigned long v)
| {
|     int i, d, w = 0;
|     u32 mask;
|
|     mask = 0xf0000000;
|     for (i = 0; mask != 0; i += 4, mask >>= 4) {
|         d = (v & mask) >> (28 - i);
|         w += putc(d + (d > 9 ? 'a' - 10 : '0'));
|     }
|     return w;
| }
+-----+
```

Azzero tutti i bit tranne
quelli associati a 'v'

da sx a dx, "f" è metà 1° byte, gli altri a 0.
nb: 1 byte rappresentato da due numeri.

Shifto di 4 verso dx (da sx a dx)

Il primo shift è di 28, poi secondo 24... in funzione di "i"

caso true

caso false, sommo 0 (da 0 a 9)

da 10 a 15 sommo 'a' - 10, ad esempio se d = 10 -> d = 10 + a - 10 = a, cioè 10 in base 16

=====

/*

vim: tabstop=4 softtabstop=4 expandtab list colorcolumn=74 tw=74

*/