ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK
Institut für
Technische Informatik und
Kommunikationsnetze

Technische Informatik 1 – Übung 5: Eingabe/Ausgabe (Computerübung)

Olga Saukh

10. & 11. November 2011

Inhalt

- **Implementierung von Device-I/O mittels Polling und Interrupts**

- **Benötigte Grundlagen**
  - **Assembler**
  - **Zustandsdiagramm**

- **Aufgaben**
  - **Bit-Test**
  - **Polling**
  - **Interrupts (Zusatzaufgabe)**

Aufgabe 1: Bit-Test: Aufgaben

- Zum Überprüfen des Status eines I/O- Devices muss oft ein einziges Bit geprüft werden. Implementieren Sie in **C *und* Assembler** dafür eine Funktion

  `int bittest(int x, int n)`

  bittest soll 1 zurückgeben, falls das Bit an der Position `n` in `x` gesetzt ist, andernfalls 0.

- Hinweis:
  - **sllv** (shift left logical variable)
    sllv $rd, $rs, $rt: $rd = $rs << $rt
  - **srlv** (shift right logical variable)
    srlv $rd, $rs, $rt: $rd = $rs >> $rt

Aufgabe 1: Bit-Test: Lösung

```c
int bittest(int x, int n) {
    if (x & (1 << n)) {
        return 1;
    }
    return 0;
}
```

```
bittest:
    li      $t0, 1
    sllv    $t0, $t0, $a1
    and     $v0, $a0, $t0
    bgtz    $v0, setone
    jr      $ra

setone:
    li      $v0, 1
    jr      $ra
```

Aufgabe 2: Polling

- **Polling** bezeichnet das regelmässige Abfragen eines Statusregisters durch ein Programm (programmierte Ein-/Ausgabe).

- In dieser Aufgabe verwenden wir Polling, um ein Zeichen vom Eingabe-Terminal zu lesen und es dann am Ausgabe-Terminal anzuzeigen (**Echo-Funktion**).

Aufgabe 2: Polling: Aufgaben

a) Zeichnen Sie ein **Zustandsdiagramm**, das die Echo-Funktion modelliert.

b) Geben Sie ein **C Programmfragment** an, das ein Zeichen vom Receiver-Device liest. Verwenden Sie Polling, um auf die Verfügbarkeit eines Zeichens zu warten.

c) Erstellen Sie ein **Assembler Programm**, das die Echo-Funktion implementiert. Verwenden Sie dazu das vorgegebene Template.

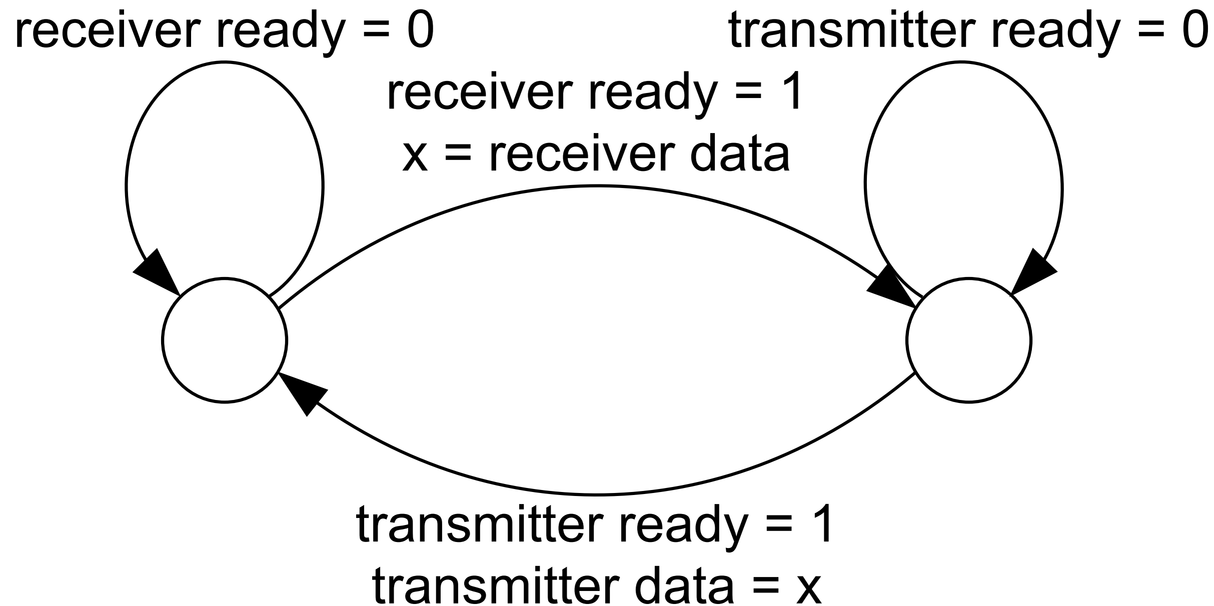
Aufgabe 2: Polling: xspim -mapped_io polling.s

```
  .text
  .globl main
main:
  addi   $sp, $sp, -4
  sw     $ra, 0($sp)
  lui    $t0, 0xffff

loop:
  #NEW CODE GOES HERE
  j loop
```

```
xspim
 PC      = 00400000   EPC     = 00000000   Cause   = 00000000   BadVAddr= 00000000
 Status  = 3000ff10   HI      = 00000000   LO      = 00000000
                          General Registers
R0  (r0) = 00000000  R8  (t0) = 00000000  R16 (s0) = 00000000  R24 (t8) = 00000000
R1  (at) = 00000000  R9  (t1) = 00000000  R17 (s1) = 00000000  R25 (t9) = 00000000
R2  (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3  (v1) = 00000000  R11 (t3) = 00000000  R19 (s3) = 00000000  R27 (k1) = 00000000
R4  (a0) = 00000000  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 10008000
R5  (a1) = 00000000  R13 (t5) = 00000000  R21 (s5) = 00000000  R29 (sp) = 7fffeffc
R6  (a2) = 00000000  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7  (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 00000000

 FIR    = 00009800   FCSR    = 00000000    FCCR    = 00000000   FEXR    = 00000000
 FENR   = 00000000
                      Double Floating Point Registers

 quit   load   reload   run   step   clear
 set value   print   breakpoints   help   terminal   mode

                               Text Segments
[0x00400000]   0x8fa40000  lw $4, 0($29)              ; 175: lw $a0 0($sp)
[0x00400004]   0x27a50004  addiu $5, $29, 4           ; 176: addiu $a1 $sp 4
[0x00400008]   0x24a60004  addiu $6, $5, 4            ; 177: addiu $a2 $a1 4
[0x0040000c]   0x00041080  sll $2, $4, 2              ; 178: sll $v0 $a0 2
[0x00400010]   0x00c23021  addu $6, $6, $2            ; 179: addu $a2 $a2 $v0
[0x00400014]   0x0c100009  jal 0x00400024 [main]      ; 180: jal main
[0x00400018]   0x00000000  nop                        ; 181: nop
[0x0040001c]   0x3402000a  ori $2, $0, 10             ; 183: li $v0 10
[0x00400020]   0x0000000c  syscall                    ; 184: syscall
[0x00400024]   0x23bdfffc  addi $29, $29, -4          ; 7: addi   $sp,$sp,-4
[0x00400028]   0xafbf0000  sw $31, 0($29)             ; 8: sw     $ra,0($sp)
[0x0040002c]   0x3c08ffff  lui $8, -1                 ; 10: lui   $t0,0xffff
[0x00400030]   0x0810000c  j 0x00400030 [loop]        ; 22: j     loop

                               Data Segments
        DATA
[0x10000000]...[0x10020000]    0x00000000

        STACK
[0x7fffeffc]                   0x00000000

        KERNEL DATA
[0x90000000]                   0x78452020  0x74706563  0x206e6f69  0x636f2000
SPIM Version 7.2.1 of August 28, 2005
Copyright 1990-2004 by James R. Larus (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
```

- Hinweis: “lw $t1, x($t0)“ lädt das Wort an Adresse 0xFFFF0000 + x nach $t1

Aufgabe 2: Polling: Lösung (a)



Aufgabe 2: Polling: Lösung (b)

```
volatile int *reg_read_ctrl = (int*)0xFFFF0000;
volatile int *reg_read_data = (int*)0xFFFF0004;
int character;

while(1) {
    int status = *reg_read_ctrl;
    if ((status & 0x1) == 0x1)
        break;
}
character = *reg_read_data;
```

Aufgabe 2: Polling: Lösung (c)

```
 .text
    .globl main

main:
    addi   $sp, $sp, -4
    sw     $ra, 0($sp)
    lui    $t0, 0xffff

loop:
poll_rx:
    lw     $t1, 0($t0)
    andi   $t1, $t1, 1
    beqz   $t1, poll_rx
    lw     $t2, 4($t0)

poll_tx:
    lw     $t3, 8($t0)
    andi   $t3, $t3, 1
    beqz   $t3, poll_tx
    sw     $t2, 12($t0)
    j      loop
```

Aufgabe 3 – Interrupts (Zusatzaufgabe)

- Ein (Hardware-) Interrupt ist ein (externes) Signal, das den normalen Programmablauf unterbricht.
- Verwendung von Interrupts:
  - Vermeidung von Polling Loops
  - Time-/Event-Driven Multi-Tasking
- In dieser Aufgabe verwenden wir Interrupts, um die Echo-Funktion zu implementieren.

Aufgabe 3 – Interrupts: Aufgaben

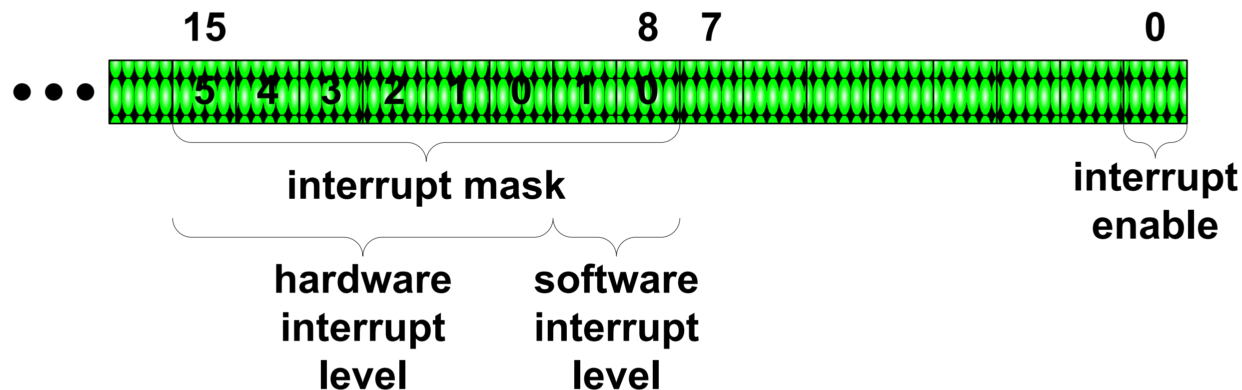
- Lesen Sie **Kapitel A.7** in Patterson/ Hennessy: “Computer Organization and Design”.

a) Schreiben Sie ein Assembler Programm, das Interrupts für das Eingabe-Terminal **aktiviert**.

b) Implementieren Sie einen **Interrupt Handler**, der das gerade eingegebene Zeichen im Ausgabe-Terminal anzeigt.

Aufgabe 3 – Interrupts: Hinweise

- mtc0 (move to coprocessor 0)
  mtc0 $rs(on processor), $rd (on coprocessor):
  $rd = $rs
  Beispiel (clear cause reg): mtc0 $0, $13
- Coprocessor 0 Status Register (Reg. Nr. 12):



- Verwenden Sie save_t0 … save_t4, um Register-inhalte während der Service-Routine zu sichern.

Aufgabe 3 – Interrupts: Lösung (a)

```
# set bit 1 of receiver control register
    li      $t4, 2
    sw      $t4, 0($t0)

# create bitmask for interrupt status
    li      $t4, 0x0801

# write status to status reg of coprocessor
    mtc0        $t4, $12
```

Aufgabe 3 – Interrupts: Lösung (b)

```
 .ktext 0x80000180
interrupt:
    .set noat
    sw     $at, save_at
    .set at

    sw     $t3, save_t3
    sw     $t2, save_t2
    sw     $t1, save_t1
    sw     $t0, save_t0

    lui    $t0, 0xffff
    lw     $t1, 4($t0)

poll:
    lw     $t2, 8($t0)
    andi   $t2, $t2, 1
    beqz   $t2, poll
    sw     $t1, 12($t0)
```

```
done:
    mtc0  $0, $13
    lw    $t0, save_t0
    lw    $t1, save_t1
    lw    $t2, save_t2
    lw    $t3, save_t3
    .set noat
    lw    $at, save_at
    .set at
    eret

    .kdata
    save_at: .word 0
    save_t0: .word 0
    save_t1: .word 0
    save_t2: .word 0
    save_t3: .word 0
```