Rechnerarchitektur Serie 4

Dominik Bodenmann 08-103-053
Orlando Signer 12-119-715

29. April 2014

1 Theorie-Teil

1.1 Aufgabe 1

Da bei beq und bne die Positionen , an die gesprungen werden soll, relativ zur aktuellen Positin angegeben wird, braucht es negative Werte.
Ebenfalls bei den Load und Store Befehlen kann der Offset zur Basisadresse negativ sein.

1.2 Aufgabe 2

Eingabe	*Ausgabe*
0xFE & 0xEF	0xEE
0xFE && 0xEF	1
0xFE | 0xEF	0xFF
0xFE || 0xEF	1
$\sim 0xFE$	-0xFF
!0xFE	0

1.3 Aufgabe 3

Mittels Schnittstellen, an denen die Hardware entsprechende Informationen zur Verfügung stellt.
Als Beispiel: Die Switches des Boards, die, je nachdem wie sie gestellt sind, ein Feld entweder mit 0 oder 1 füllen.

1.4 Aufgabe 4

1. Singlecycle-Implementation Das Resultat einfach invertieren, falls die ALU beim Vergleich 0 zurück gibt. (NOT-Gatter am 0-Ausgang der ALU)

2. Multicycle-Implementation Analog Singlecycle-Implementation.

1.5 Aufgabe 5

Um die Daten, die für den nächsten Rechenschritt gebraucht werden, zwischenzuspeichern, da die nächste Berechnungsstufe möglicherweise noch nicht zur Verfügung steht.

1.6 Aufgabe 6

Structural Hazard: Tritt auf, falls während einem Clock-Cycle die gleiche Resource von zwei verschiedenen Instruktionen benötigt wird. Z.B. wenn eine Instruktion im 4. Schritt etwas vom Memory liest, gleichzeitig aber eine andere Instruktion in der Pipeline ihre Instruktion vom Memory liest. Dies kann umgangen werden, indem es separate Memories für Instruktionen und Daten gibt.
Data Hazard: Tritt auf, wenn eine Instruktion noch nicht ausgeführt werden kann, da sie vom Resultat einer anderen Instruktion abhängt. Z.B. eine Instruktion berechnet einen Wert und schreibt ihn in Register $1. Benötigt die nächste Instruktion den Wert von $1, so muss sie in der Pipeline warten, bis die erste Instruktion den Wert ins Register geschrieben hat. Mittels Forwarding kann dieses Problem umgangen werden. Dabei dient der berechnete Wert von der ALU direkt als Input für die ALU. Mittels Mux kann dann vor der ALU entschieden werden, welcher Wert übernommen wird.
Control Hazard: Tritt bei Instruktionen auf, die den Befehlszäher verändern (PC != PC + 4), z.B. bei Branching oder Jumps. Dabei sind bei nach einem Branch/Jump eventuell falsche Instruktionen in der Pipeline, die geflusht werden müssen. Deshalb ist es wichtig, die Zieladresse so früh wie möglich zu berechnen, damit nicht zu viele unnötige Instruktionen in der Pipeline geflusht werden.

1.7 Aufgabe 7

Auf Folie 19 werden die Resultate aus dem 4. Cycle der beq Instruction bereits im ersten Cycle der lw instruction gebraucht.
Auf Folie 15 ist dies nur mit Abstand 2 der Fall.

1.8 Aufgabe 8

add $t1, $t2, $t3
sub $t4, **$t1** (Forwarding), $t2
lw $s2, **200($t1)** (Forwarding)
add $s3, $t1 ,**$s2** (Stall)

2 Praktischer Teil

Listing 1: knightRider.s

```
/* TODO: Task (b) Please fill in the following lines, then remove this line.
 *
 * author(s):   Dominik Bodenmann
 *       Orlando Signer
 * modified:    2014-04-29
 *
 */

.include "nios_macros.s"
.include "address_map.s"

/*****************************************************************************
 * TEXT SECTION
 */
.text

/*****************************************************************************
 * Entry point.
 */
.global _start
_start:
  /* set up sp and fp */
  movia   sp, 0x007FFFFC      # stack starts from largest memory address
  mov     fp, sp

  /* This program exercises a few features of the DE1 basic computer.
   *
   * It performs the following:
   *     1. displays a red light wandering from LEDR0 to LEDR9 and back again
        (and so on...)
   *     2. speed of light can be increased by KEY3, decreased by KEY1 and
       initial value can be restored by KEY2
   */

  /* set up timer interval = 0x0000C350 steps * 1/(50 MHz) = 1 millisecond*/
  movia r15, TIMER_COUNTER_LOW
  movui r16, 0xC350
  sthio   r16, 0(r15)

  movia r15, TIMER_COUNTER_HIGH
  movui r16, 0x0000
  sthio   r16, 0(r15)

  /* start interval timer, enable its interrupts */
  movia r15, TIMER_STOP_START_CONT_ITO
  movi    r16, 0b0111   # START = 1, CONT = 1, ITO = 1
  sthio   r16, 0(r15)
```

```
46
47    /* enable pushbutton interrupts */
48    movia r16, PUSHBUTTON_BASE
49    movi    r15, 0b01110    # set all 3 interrupt mask bits to 1 (bit 0 is Nios
          II Reset)
50    stwio   r15, 8(r16)
51
52    /* enable processor interrupts */
53    movi    r16, 0b011      # enable interrupts for timer and pushbuttons
54    wrctl   ienable, r16
55    movi    r16, 1
56    wrctl   status, r16
57
58    /* r15 stores the memory address for the LEDs
59     * r17 stores the active LED
60     * r18 the direction to move (0 -> downwards, 1 -> upwards)
61     * r19 stores the delay-time
62     * r20 and r21 are for temporary use
63     */
64
65    /* Initialize first red LED (light up) */
66    movia   r15, RED_LED_BASE
67    movi    r17, 0x1    # Bitmask for first LED
68    movi    r18, 0x0    # Direction bit (will be inverted first)
69    movia   r19, 0xC8   # Wait 0xC8 = 200 msecs
70
71
72    CHECK_BORDER:
73    movi    r20, 0x1          # Set value for lowest bit
74    beq     r17, r20, INVERT_DIRECTION  # If we are on the lowest bit, invert
          direciton
75    movi    r20, 0x200          # Set value for highest bit
76    beq     r17, r20, INVERT_DIRECTION  # If we are on the highest bit, invert
          direction
77
78    CHECK_DIRECTION:
79    movi    r20, 0x1
80    beq     r18, r20, MOVE_UP       # Check the direction we move, MOVE_UP if
          the move-bit is 1
81    br      MOVE_DOWN         # else MOVE_DOWN
82
83    MOVE_UP:
84    stwio   r17, 0(r15)         # Write LED-Bitmask to memory
85    slli    r17, r17, 0x1       # left-shift the value by one
86    br      DELAY           # Delay the next output
87
88    MOVE_DOWN:
89    stwio   r17, 0(r15)         # Write LED-Bitmask to memory
90    srai    r17, r17, 0x1       # right-shift the value by one
91    br      DELAY           # Delay the next output
92
93    DELAY:
94    movia   r20, TIME
95    ldwio   r21, 0(r20)         # Get the Time from the Counter
```

```
96    blt     r20, r19, DELAY       # Check if we waited more than (r19) steps (
          go to DELAY else)
97    stwio   r0, 0(r20)          # Reset Time counter
98    br      CHECK_BORDER        # Go to check the borders (--> start again)
99
100   INVERT_DIRECTION:
101   xori    r18, r18, 0x1       # Invert the direction
102   br      CHECK_DIRECTION
103
104
105
106
107 /*******************************************************************************

108   * DATA SECTION
109   */
110 .data
111
112 /* to count how much time has passed*/
113 .global TIME
114 TIME:
115   .word 0
116
117 /* TODO: Task (c) you may also want to add things here (but you don't need to
        ) */
118 .end
```

Listing 2: pushbutton.s

```
/* TODO: Task (b) Please fill in the following lines, then remove this line.
 *
 * author(s):   Dominik Bodenmann
 *       Orlando Signer
 * modified:    2014-04-29
 *
 */

.include "nios_macros.s"
.include "address_map.s"


/*****************************************************************************
 * Pushbutton - Interrupt Service Routine
 *
*****************************************************************************/

.global PUSHBUTTON_ISR
PUSHBUTTON_ISR:

  /*
   * r19 stores the delay time
   * r20 stores the button pressed values
   * r21 is for temporary use
   */

  movia   r21, PUSHBUTTON_BASE  # Memory address for pushed buttons
  ldwio   r20, 0xC(r21)     # read register with values of buttons
  stwio   r0, 0xC(r21)      # reset the pushed buttons

  movia   r21, 0b100
  beq     r20, r21, DEFAULT   # Check if KEY2 is pressed
  movia   r21, 0b1000
  beq     r20, r21, FASTER    # Check if KEY3 is pressed
  movia   r21, 0b10
  beq     r20, r21, SLOWER    # Check if KEY1 is pressed
  ret

  DEFAULT:
  movia   r19, 0xC8       # Set the default delay
  ret

  FASTER:
  subi    r19, r19, 0x32      # Subtract 0x32 (50) msecs to delay
  ret

  SLOWER:
  addi    r19, r19, 0x32      # Add 0x32 (50) msecs to delay
  ret

.end
```