Technische Informatik I

Uebung 2

Simon Umbricht, Patrick Boenzli

https://dev.orxonox.net/wiki/boenzlip/tinf1

Exercise 2) – Basisbloecke

Ziel: Aufteilung des Codes in logische Blöcke, die sequentiell abgearbeitet werden (d.h. ohne Jumps und Branches)

Exercise 2) – Basisbloecke

```
00000000 <matrix>:
   0:   8fac0010        lw      t4,16(sp)
   4:   18e00029        blez    a3,ac <matrix+0xac>
   8:   00000000        nop
   c:   00007821        move    t7,zero
  10:   08000022        j       88 <matrix+0x88>
  14:   00000000        nop
  18:   010c0018        mult    t0,t4
  1c:   00001012        mflo    v0
  [...]
  48:   25290004        addiu   t1,t1,4
  4c:   14e8fff2        bne     a3,t0,18 <matrix+0x18>
  50:   00000000        nop
  54:   adaa0000        sw      t2,0(t5)
  58:   256b0001        addiu   t3,t3,1
  [...]
```

Neuer Block beginnt:

- bei jedem angesprungenen Label (in diesem Fall auch 0x18)
- nach jedem jump / branch Befehl. Die nachfoldende Anweisung zählt noch zum alten Block dazu (Branch Delay)

Exercise 2) – Basisbloecke

```
00000000 <matrix>:
   0:   8fac0010        lw      t4,16(sp)
   4:   18e00029        blez    a3,ac <matrix+0xac>
   8:   00000000        nop
   c:   00007821        move    t7,zero
  10:   08000022        j       88 <matrix+0x88>
  14:   00000000        nop
  18:   010c0018        mult    t0,t4
  1c:   00001012        mflo    v0
  [...]
  48:   25290004        addiu   t1,t1,4
  4c:   14e8fff2        bne     a3,t0,18 <matrix+0x18>
  50:   00000000        nop
  54:   adaa0000        sw      t2,0(t5)
  58:   256b0001        addiu   t3,t3,1
  [...]
```

Neuer Block beginnt:
- bei jedem angesprungenen Label (in diesem Fall auch 0x18)
- nach jedem jump / branch Befehl. Die nachfoldende Anweisung zählt noch zum alten Block dazu (Branch Delay)

Exercise 2) – Basisbloecke

```
00000000 <matrix>:
   0:   8fac0010        lw      t4,16(sp)
   4:   18e00029        blez    a3,ac <matrix+0xac>
   8:   00000000        nop
   c:   00007821        move    t7,zero
  10:   08000022        j       88 <matrix+0x88>
  14:   00000000        nop
  18:   010c0018        mult    t0,t4
  1c:   00001012        mflo    v0
  [...]
  48:   25290004        addiu   t1,t1,4
  4c:   14e8fff2        bne     a3,t0,18 <matrix+0x18>
  50:   00000000        nop
  54:   adaa0000        sw      t2,0(t5)
  58:   256b0001        addiu   t3,t3,1
  [...]
```

Neuer Block beginnt:
- bei jedem angesprungenen Label (in diesem Fall auch 0x18)
- nach jedem jump / branch Befehl. Die nachfoldende Anweisung zählt noch zum alten Block dazu (Branch Delay)

Exercise 2) – Inhalt der Register

Argument-Register

a0	&A
a1	&B
a2	&C
a3	n

Stack

$sp + 16	m

Temporäre Register

t0	k
t1	&A[...] = &A + offset
t2	sum
t3	j
t4	m (wird vom Stack geladen)
t5	&C[...] = &C + offset
t6	4*i*m = &C[...] - &C = offset in C
t7	i

Resultate-Register

v0	&B[...] = &B + offset. v0 hier **nicht** return value (void)!

Loesungen

Exercise 1)

```
    move v0, zero                    /* Resultat initial. */

skalar_loop:
    beq a2, zero, skalar_end_loop /* Abbruchbedingung */

    sub a2, a2, 1                    /* Laufvar. anpassen */
    lw t0, 0(a0)                     /* A[i] laden */
    lw t1, 0(a1)                     /* B[i] laden */

    addi a0, a0, 4                   /* A, B auf nächstes */
    addi a1, a1, 4                   /* Element setzen */

    mult t0, t1                      /* A[i]*B[i] rechnen, */
    mflo t2                          /* untere 32 Bits laden */

    add v0, v0, t2                   /* Zum Resultat addieren */
    j skalar_loop                    /* nächste Zeile in A, B*/

skalar_end_loop:
    j ra
```

Exercise 2)

```
a0=A,
a1=B
a2=C
a3=n
t4=m (via stack übergeben)

t0=k
t1=A[i*n] (beim Eintritt in den inneren Loop)
t2=sum
t3=j
t4=m
t5=C[i*n]
t6=4*i*n
t7=i

00000000 <matrix>:
   0:   lw      t4,16(sp)              // m = *(sp+16)
```

Exercise 2)

```
    4:   blez    a3,ac <matrix+0xac>   // if (n<=0) goto end

    8:   nop
    c:   move    t7,zero               // i = 0
   10:   j       88 <matrix+0x88>      // goto testm

   14:   nop
inner:
   18:   mult    t0,t4
   1c:   mflo    v0                    // v0 = k*m
   20:   addu    v0,t3,v0              // v0 = k*m+j
   24:   sll     v0,v0,0x2             // v0 = 4*(k*m+j)
   28:   addu    v0,v0,a1              // v0 = 4*(k*m+j)+B (Adr von B[k*m+j])
   2c:   lw      v1,0(t1)              // v1 = A[i*n(+k)] (vgl Instr. 48)
   30:   lw      v0,0(v0)              // v0 = B[k*m+j]
   34:   nop
   38:   mult    v1,v0
   3c:   mflo    v1                    // v1 = A[i*n+k]*B[k*m+j]
   40:   addu    t2,t2,v1              // sum = sum + A[i*n+k]*B[k*m+j]
   44:   addiu   t0,t0,1               // k++
   48:   addiu   t1,t1,4               // erhöhe k in A[i*n+k] um 1
   4c:   bne     a3,t0,18 <matrix+0x18>// if (n!=k) goto inner
```

Exercise 2)

```
  50:   nop
  54:   sw      t2,0(t5)                // C[i*n+j] = sum
  58:   addiu   t3,t3,1                 // j++
  5c:   addiu   t5,t5,4                 // erhöhe j in &C[i*n+j] um 1
  60:   beq     t4,t3,7c <matrix+0x7c>// if (j=m) goto check_finish_outer

  64:   nop
initialize_inner_loop:
  68:   addu    t1,a0,t6                // t1 = a[i*n]
  6c:   move    t0,zero                 // k = 0
  70:   move    t2,zero                 // sum = 0
  74:   j       18 <matrix+0x18>        // goto inner

  78:   nop
check_finish_outer:
  7c:   addiu   t7,t7,1                 // i++
  80:   beq     a3,t7,ac <matrix+0xac>// if(i==n) goto end

  84:   nop

  88:   blez    t4,7c <matrix+0x7c>     // if(m<=0) goto check_finish_outer
```

Exercise 2)

```
compute_result_addr
  8c:   nop
  90:   mult    t7,a3
  94:   mflo    v0                      // v0 = i*n
  98:   sll     t6,v0,0x2               // t6 = 4*i*n (Wordaddresse)
  9c:   addu    t5,a2,t6                // t5 = &C[i*n] (Adr von C[i*n])
  a0:   move    t3,zero                 // j = 0
  a4:   j       68 <matrix+0x68>        // goto initialize_inner_loop

  a8:   nop
end:
  ac:   jr      ra                      // return to caller

  b0:   nop
```