Semestrální práce z předmětu KIV/UPA

Jan Bařtipán / A03043
bartipan@studentes.zcu.cz

Zadání

Program přečte ze vstupu dvě čísla v hexadecimálním tvaru a vypíše jejich součet (opět v hexadecimální tvaru).

Rozbor problému

Převod vstupního řetězce na číslo

O konverzi ze vstupního řetězce na číslo se stará rutina read_num. Její princip lze pospat takto:

```
var input: array[1..8] of char;
     i,cislo,cislice: integer;
cislo := 0;
for i := 1 to 8 do begin
      if input[i] < '0' then goto error;
      if input[i]<='9' then cislo := input[i] -'0';
      else begin
            if input[i] < 'a' then goto error;
            if input[i] <= 'f' then cislice := input[i] -'a' + 10;
      end;
      if i < 8 then cislo := cislo shl 8;
      cislo := cislo + cislice;
end;
```

Převod čísla na výstupní řetězec

Obdobným způsobem jako v předchozím případě lze udělat obrácenou konverzi. Pro úplnost uvedu její algoritmus:

```
var ouput: array [1..8] of char;
  cislo, cislice;
for i := 8 downto 1 do begin
      cislice := cislo and 0fh;
      if cislice < 10 then output[i] := cislice + '0';
      else output[i] := (cislice -10) + 'a';
      cislo := cislo shr 8;
end;
```

Nahrazování pseudoinstrukcí

Překladač (resp. simulátor) pro pohodlí programátora rozšiřuje instrukční sadu o další instrukce, které pří překladu (resp. simulaci) nahrazuje jednou či více instrukcemi ze sady procesoru.

1) Například pseudoinstrukce `move $t0, $a0` je nahrazena instrukcí `addu $t0, $0, $a0` (registr $0 je speciálním registrem, který vždy obsahuje nulu).
2) Další pseudoinstrukcí je `subu $t0, $t0, 1`. Tu překladač (simulátor) nahradí takto: `addiu $t0, $t0, -1`.
3) Jako poslední zde zmíním o větvící instrukci. Například `bgt $t0, 57, chck_af` se nahradí za tuto posloupnost instrukcí:

```
  slti $1, $8, 58 ($t1 = $8, $at = $1)
  beq $1, $0, 12
```

Datový hazard

Potenciální datový hazard nastává v případech, kdy se instrukce pokusí přečíst obsah paměti po instrukci, která do tohoto zdroje zapisuje. Například:

```
        move    $ra, $t0    # restore return address from t0
        jr      $ra                 # return
```

Listing programu

Úsek programu, kde může dojít k datovému hazardu je vyznačen kurzívou a podtržením. Úseky, kde dochází ke zpoždění čtení z paměti či zpoždění skokových instrukcí jsou označeny tučně.

```
                .data
instr:          .asciiz "12345678"
var1:           .word   0x01
var2:           .word   0x02
var3:           .word   0x00
ret:            .word   0x00
errstr:         .asciiz "\nError: Wrong number format. Accepted are [0-9a-f]{8}\n"
err2str:        .asciiz "\nError: Empty string is not number ;)\n"
msg1:           .asciiz "\n   "
msg2:           .asciiz "\n + "
msg3:           .asciiz "\n-----------\n   "
newlinestr:     .asciiz "\n"
outstr:         .asciiz "12345678"


        .text
        .globl main

main:
        nop

        li      $v0, 4
        la      $a0, msg1
        syscall

        jal     read_num            # read first nubmer
        nop
        sw      $a0, var1           # store it into var1
        nop

        li      $v0, 4
        la      $a0, msg2
        syscall

        jal     read_num            # read second number
        lui     $a0, 0x344
        sw      $a0, var2           # store it into var2

        lw      $t0, var1           # load first number into t0
        lw      $t1, var2           # load second number into t1
        nop
        nop
        addu    $t2, $t0,$t1        # t2 = t0 + t1

        sw      $t2, var3           # store result into var3

        li      $v0, 4
        la      $a0, msg3
        syscall
```

```
        lw     $a0, var3
        jal    print_hex
        jal    newline

        lw     $ra, ret           # restore global return address
        nop
        nop
        j      $ra                # quit
        nop
        nop



read_num:                         # read number function
        addiu $sp, $sp, -32       # push stack
        sw    $ra, 20($sp)        # store return address to stack
        sw    $fp, 16($sp)        # store frame pointer to stack
        addiu $fp, $sp, 28        # setup new frame pointer

        li    $v0, 8              # read string (of 8 char)
        la    $a0, instr          # ... into [instr]
        li    $a1, 9              # ... maximal length of string to read is 8
        syscall

        li     $t2, 0             # t2 sotres value of number
        la     $a0, instr         # a0 now points to beginig of read string



        lb    $t0, 0($a0)         # load first byte of string into t0
        nop
        nop
        beq   $t0, 0, error2      # empty string -> error
        nop
        nop

conv_loop:                        # for all characters in read string

chck_09:                          # check range [0-9]
        blt   $t0, 48, error      # t0 < '0' -> error
        nop
        nop
        bgt   $t0, 57, chck_af    # t0 > '9' ->  check range [a-f]
        nop
        nop
        addiu $t1, $t0, -48       # t1 = digit value of t0
        j     chck_ok
        nop
        nop

chck_af:                          # check range [a-f]
        blt   $t0, 97, error      # t0 < 'a' -> error
        nop
        nop
        bgt   $t0, 102, error     # t0 > 'f' -> error
        nop
        nop
        addiu $t1, $t0, -87       # t1 = digit value of t0

chck_ok:                          # range test OK

        addu  $t2, $t2, $t1       # t2 += t1

        addu  $a0, $a0, 1         # go to next charcter
        lb    $t0, 0($a0)         # load next character into t0
```

```
        nop
        nop
        beqz  $t0, conv_loop_end  # is end of string -> end_loop
        nop
        nop
        mul   $t2, $t2, 16        # t2 <<= 8
        j     conv_loop           # is not end of string -> loop
        nop
        nop
conv_loop_end:
        lw    $ra, 20($sp)        # restore return address from stack
        nop
        lw    $fp, 16($sp)        # restore frame pointer from stack
        addiu $sp,$sp, 32         # pop stack
        move  $a0, $t2            # store result in a0
        jr    $ra                 # end of read_num

error:                            # Prints error message 1
        la      $a0, errstr       # set address of error message to a0
        j       error_print       # go to print it
        nop
        nop

error2:                           # Prints error message 2
        la      $a0, err2str      # store address of error message to a0

error_print:
        li      $v0, 4            # print error message
        syscall

        li     $v0, 20            # ... and exit
        syscall



newline:
        move   $t0, $ra           # store return address into t0
        la     $a0, newlinestr    # a0 points to new line string
        li     $v0, 4             # number of system call
        syscall                   # print it
        move   $ra, $t0           # restore return address from t0
        jr     $ra                # return
        nop
        nop
        nop

print_hex:
        addiu  $sp, $sp, -32
        sw     $ra, 24($sp)
        sw     $fp, 16($sp)
        addiu  $fp, $fp, 28

        move   $t0, $a0           # t0 stores number to print
        la     $a0, outstr        # a0 stores pointer to be stored actual digit
        addu   $a0, $a0, 7        # ... now it pointes to last character of buffer
        li     $t1, 8             # t1 loop counter
        li     $t2, 0x00f         # t2 mask to get last number

print_hex_loop:
        and    $t3, $t0, $t2      # t3 = t0 % 16

                                  # convert from digit number to character
        bgt    $t3, 0x09, hex_conv_af  #<--+
        nop                       #    |
        nop                       #    |
hex_conv_09:                      #    |
```

```
        addu    $t3, $t3, 48            #   |
        j       hex_conv_ok             #   |
        nop
        nop
hex_conv_af:                            #   |
        addu    $t3, $t3, 87            #   |
hex_conv_ok:                            #<--+
        nop
        nop
        sb      $t3, 0($a0)         # store t3 into buffer
        addiu   $t1, $t1, -1        # t1--
        addiu   $a0, $a0, -1        # a0--
        div     $t0, $t0, 0x10      # t0 /= 16
        bnez    $t1, print_hex_loop
        nop
        nop

        li      $v0, 4              # print buffer
        la      $a0, outstr
        syscall

        lw      $ra, 24($sp)
        lw      $fp, 16($sp)
        addiu   $sp, $sp, 32
        jr      $ra
```

Závěr

K vývoji a ladění programu byl použit simulátor MIPS procesoru PCSim ve verzi 7.0 ve verzi pro MS Windows. Pokoušel jsem s e zprovoznit stejný simulátor i na (mojí domovské) platformě GNU/Linux. Ačkoliv se mi povedlo po urputných bojích program zkompilovat a nainstalovat, vykazoval známky abnormálního chování, a proto jsem raději používal Windowsovou verzi.

Binární forma programu je ke stažení na adrese: http://zcu.cz/~bartipan/upa/sum.s.