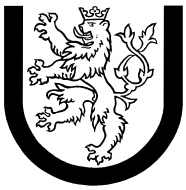

ZÁPADOČESKÁ
UNIVERZITA
V PLZNI

Úloha 2

z předmětu

Úvod do počítačových architektur

Jméno a příjmení: *Martin Lipinský*
Osobní číslo: ***A05450***
Studijní skupina: *Dálkové studium*
Obor: *INIB-INFB*
E-mail: *martin@lipinsky.cz*

Označení zadání: *Výpočet faktoriálu*

Datum odevzdání: 18.1.2008

Obsah

1 Zadání

Napište v assembleru pro procesor MIPS R3000 program, který spočítá faktoriál zadaného čísla. Číslo zadejte jako konstantu přímo v programu. Kód odlaďte na simulátoru SPIM se zapnutými volbami „-delayed_branches -delayed_loads“. Simulujte tak zpožděný výsledek některých instrukcí.

2 Popis řešení

2.1 Algoritmus v C

Pro připomenutí algoritmu výpočtu faktoriálu jsem si napřed program napsal v jazyce C. To mi usnadnilo následné programování problému v assembleru. Umožnilo mi to soustředit se na specifika MIPSu a assembleru.

```
/*
Program pro vypocet faktorialu rekurzivme martin@lipinsky.cz
Na libovolne linux platforme:
Preklad: gcc fakt.c -o fakt -ansi -pedantic -Wall
Spusteni: ./fakt
*/

#include <stdio.h>

/* funkce vracejici rekurivnim volanim faktorial */
int faktorial(int n) {
        if (n <= 1) return 1;
        return n * faktorial(n - 1);
}

int main(int argc, char *argv[]) {
        int i;
        /* kolik faktorialu chceme spocitat */
        int MAX=10;
        /* smycka pro vypis faktorialu od nuly do 10 */
        for (i=0; i<=MAX; i++) {
                printf("Vypocet faktorialu: %d!=",i);
                printf("%d \n",faktorial(i));
        }
        /* vracime nulu */
        return 0;
}
```

2.2 Programování pro MIPS

Požadavek zněl aby program byl funkční i při emulaci zpoždění čtení dat z paměti (2 instrukce) a skoku (provede ještě jednu instrukci za skokem). To lze ošetřit buď vhodným řazením

instrukcí, nebo vložením prázdné instrukce nop. Tomu jsem se snažil vyhnout ale ne všude se mi to podařilo.

2.3 Kód programu v assembleru

Vlastní kód přikládám níže. Byl odladěn v Linuxu. Krokován v programu „**xspim -delayed_branches -delayed_loads fakt.asm**“ který umožňuje sledovat obsah registů a paměti. Závěrečné ladění už proběhlo jen v konzolovém „**spim**“ který na pozadí spustí program a prezentuje pouze konzolové výstupy.

```
        .data                           # datova část

#fact:  .word   0x01                    # cislo jehoz chceme faktorial
#fact:  .word   0x02                    # cislo jehoz chceme faktorial
fact:   .word   0x06                    # cislo jehoz chceme faktorial
koment: .asciiz "! je:"                 # napis
enko:   .asciiz "\n"                    # odradkovani

        .text                           # programova cast
main:
        lw      $t0,fact                # V t0 bude faktorial
        move    $t4,$ra                 # v t4 schovat navratovou adresu
        move    $t1,$t0                 # V t1 budeme mit N-1
        move    $t2,$t0                 # v t2 si budeme drzet N
        jal     facto                   # skok na faktorial, move se jeste provede


facto:
        bne     $t1,1,rec               # podminka ukonceni rekurze
        jal     print                   # navrat
        nop

rec:
        addi    $t1,$t1,-1              # vypocet  N-1
        mul     $t0,$t1,$t0             # n=n*facto(n-1)
        jal     facto
        nop

print:
        li      $v0,1                   # sluzba tisku intu
        move    $a0,$t2                 # k tisku pripravit vysledek scitani
        syscall                         # proved

        li      $v0,4                   # tisk stringu
        la      $a0,koment              # adresa kde je string do a0
        syscall

        li      $v0,1                   # sluzba tisku intu
        move    $a0,$t0                 # k tisku pripravit vysledek scitani
        syscall                         # proved

        li      $v0,4                   # tisk stringu
        la      $a0,enko                # adresa kde je string do a0
        syscall

        j       $ra                     # return, skok se provede az po move
        move    $ra,$t4                 # obnovit navratovou adresu, az za skokem protoze se
                                        # stejne provede :)
```

3 Závěr

Programování R3000 mě seznámilo se základy RISC architektury a s důsledků které z ní plynou pro programátora a nebo překladače vyšších jazyků. Prezentovaný program by šel jistě dále vylepšit tak aby byl efektivnější (odstranit i další nop instrukce z podprogramů).