Universita Pardubice
Fakulta elektrotechniky a informatiky

Mikroprocesorová technika

Semestrální práce

Jméno: Chmelař Pavel
Datum: 14. 5. 2008

Úkol:

Příklad č. 1

V paměti dat je uložen blok 8 b čísel se znaménkem. Sestavte program, který tato čísla srovná sestupně podle velikosti. Pro řazení použijte algoritmus Select Sort. Řazení bude napsáno jako podprogram se dvěma parametry – adresa začátku bloku dat a jeho délka.

Sellect short:

Principem tohoto řazení je výběr mezního prvku (maximum nebo minimum) z tříděné posloupnosti a jeho záměna s prvním (posledním) prvkem. V dalším kroku máme pole o (n-1) prvcích a opakujeme totéž. Takto postupujeme až do úplného seřazení posloupnosti. Získaná posloupnost bude seřazená vzestupně v případě, že budeme vybírat z nesetříděné posloupnosti nejmenší prvek a zařadíme ho na začátek posloupnosti, a sestupně v případě, že vybereme největší prvek z nesetříděné posloupnosti a zařadíme ho na začátek. Při tomto řazení se vlastně setříděná část pole postupně zvětšuje a nesetříděná část naopak zmenšuje. Algoritmus končí v případě, že nesetříděná část již neobsahuje žádný prvek. Tato metoda je vnitřní, přímá a jednoduchá. Nevýhodou tohoto algoritmu je, že jestliže se na vstupu nachází již částečně setříděná posloupnost, algoritmus to nerozlišuje a proběhne v maximálním počtu koků.

Princip činnosti – řazení sestupně, čísla se znaménkem ve dvojkovém doplňku:

1. V poli nalezneme prvek s největší hodnotou a zapamatujeme si jeho pořadí
2. Vyměníme tento prvek s prvkem na prvním místě
3. Postupně opakujeme body 1. a 2. nad zbývající, nesetříděnou částí pole

Složitost:

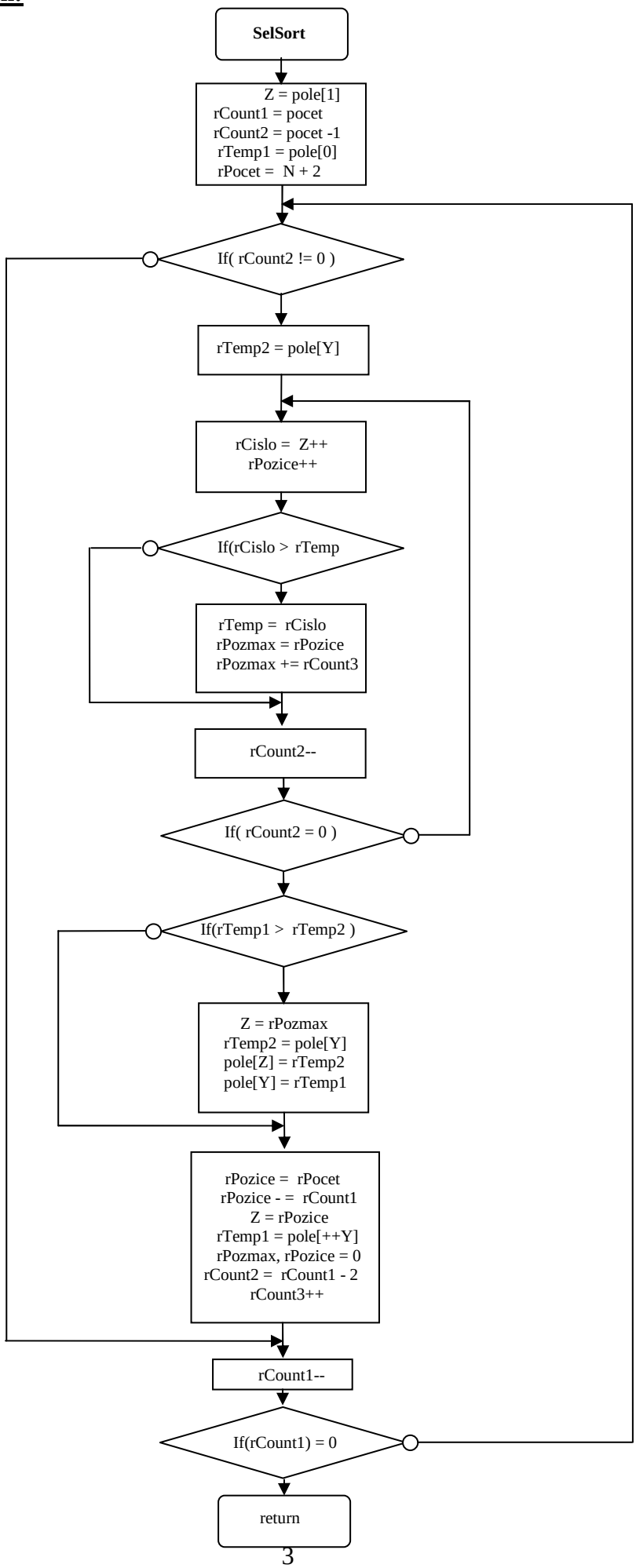
Složitost algoritmu SelectSort je $O(n^2)$.

Implementace:

Použití:

1) Z paměti programu se nahrají čísla k srovnání do paměti dat
2) Před voláním metody se musí nahrát parametr délky pole do příslušného registru a pointer Y se musí inicializovat na začátek pole v paměti (pole[0])
3) Zavolat podprogram SelSort

Vývojový diagram:



Tabulka použitých registrů:

Registr:	**Pojmenování:**	**Funkce:**
r16	rTemp1	pomocný registr v metodě
r17	rTemp2	pomocný registr v metodě
r18	rPozice	počítadlo pozice
r19	rPozmax	pozice prvku maximální velikosti
r20	rCount1	počítadlo hlavního cyklu
r21	rCount2	počítadlo vedlejšího cyklu
r22	rCount3	počítadlo hlavního cyklu + výpočet pozice prvku
r23	rCislo	pomocný registr v metodě (vnořený cyklus)
r24	rPocet	parametr délky pole

Slovní popis algoritmu:

1) Po zavolání metody se pointer Z nastaví na adresu pole[1], rCounter2 pro vnořený cyklus se nastaví na hodnotu počet – 1 prvků pole, rCounter1 se nastaví na hodnotu počtu prvků pole a do rTemp1 se uloží hodnota pole[1]. rPozice se nastaví na N+2.

2) Následuje test hodnoty rCount2, která se porovná s nulou a jestliže je tato hodnota větší, tak se spustí vnořený cyklus, jinak se skočí na konec celé metody, protože nejsou prvky k srovnání.

3) Těsně před vnořeným cyklem se nejprve zapíše do rTemp2 hodnota prvku pole, na které ukazuje pointer Y, poté se už v cyklu do rCislo uloží hodnota pole, na kterou ukazuje pointer Z a následně se zvětší o jedničku a rPozice se také zvětší o jedničku.

4) Dále se porovnají hodnoty rCislo s rTemp1 a jestliže je rCislo větší, poté se zapíše do rTemp jeho hodnota a do rPozmax se zapíše rPozice + rCount3. Dekrementuje se hodnota rCount2, proces se opakuje, dokud se neprojde celé zbývající pole.

5) Nyní následuje porovnání hodnot rTemp1 s rTemp2 a jestliže se tyto hodnoty od sebe neliší, přeskočí se „prohazovací“ proces mezi prvky, jinak se pokračuje dál na výměnu prvků

6) Pointer Z se nastaví na pozici maximálního prvku v poli, do rTemp2 se zapíše prvek pole, na které ukazuje pointer Y a rTemp2 se zapíše na pozici pointru Z, rTemp1 na pozici Y

7) Následuje procedura přípravy na další cyklus, tzn. pozice pointru Y se zvýši o jedničku, jeho hodnota se zapíše do rTemp1. Pomocí rCount1 se vypočítá hodnota rCount2 a pointru Z, aby ukazoval o jednu adresu v paměti výš, než pointer Y. Zvětší se o jedničku rCount3.

8) Nakonec se porovná hodnota rCount1 s nulou, jestliže je hodnota větší jak nula, cyklus se opakuje od bodu 2), jinak je pole srovnané a navrací se zpět do hlavního programu

Rozbor „fint“:

1) Před vnořeným cyklem se do rTemp2 zapíše hodnota prvku pole, na které ukazuje pointer Y a po projití vnořeného cyklu se tato hodnota porovná s hodnotou rTemp1, do které se ve vnořeném cyklu ukládá hodnota největšího prvku pole. Jestliže jsou stejné tak se přeskočí vyměňovací procedura, tím se ušetří čas a třídění probíhá značně kratší dobu, než kdyby se stejné prvky musely nesmyslně mezi sebou vyměnit.

   Díky tomuto porovnání obecně platí, čím je pole více srovnané, tím menší dobu trvá třídění. Jedinou nevýhodou tohoto principu je, že když se přerovnávají všechny prvky v poli, trvá celé třídění o něco déle než bez použití této testovací podmínky.

2) Registr pojmenovaný rCount3 slouží jako pomocné počítadlo, které čítá, kolikrát proběhl základní cyklus a toho se využívá při výpočtu pozice, na které se nachází prvek s nejvyšší hodnotou. Pole, které se prochází, se postupně zmenšuje, protože není třeba kontrolovat srovnané prvky. Proto se ve vnořeném cyklu k rPozmax (maximální prvek zmenšeného pole) přičte tato hodnota rCount3 a výsledkem je správná pozice prvku v rozsahu celého pole.

3) Registr pojmenovaný rPozice slouží, k počítání pozice, kde se nacházíme v postupně zmenšujícím poli ve vnořeném cyklu vlivem třídění. Jestliže je hodnota rCislo větší než rTemp, dojde k zápisu rCislo do rTemp a rPozice ukazuje pozici nejvyššího prvku zmenšeného pole, následně se tedy zapíše do rPozmax.

4) Celá metoda funguje na principu, že pointer Y ukazuje zpočátku na první prvek pole (pole[0]) a pointer Z ukazuje vždy na prvek pole o jeden větší. Ve vnořeném cyklu se projde pole, od pozice, kam ukazuje pointer Z, až dokonce a najde se nejvyšší prvek, zapamatuje se jeho pozice, díky této získané pozici se nastaví pointer Z na hodnotu nejvyššího prvku v poli. Poté dojde výměně mezi menším a větším prvkem. Následně se na další cyklus připraví pointer Y, aby ukazoval na prvek o jednu hodnotu větší než v předešlém cyklu a pomocí rPocet, rCount1 se dopočítá pozice pointru Z, aby ukazoval na prvek o jednu větší než pointer Y.

Program:

```
.include "m32def.inc"          // konstanty ATmega32

//konstanty
.equ    N = 6                  // Pocet prvku pole

//registry
.def    rTemp1 = r16           // pomocny registr v metode
.def    rTemp2 = r17           // pomocny registr v metode
.def    rPozice = r18          // pocitadlo pozice
.def    rPozmax = r19          // pozice prvku maximalni velikosti
.def    rCount1 = r20          // pocitadlo hlavniho cyklu
.def    rCount2 = r21          // pocitadlo vedlejsiho cyklu
.def    rCount3 = r22          // pocitadlo hlavniho cyklu + vypocet pozice prvku
.def    rCislo = r23           // pomocny registr v metode (vnoreny cyklus)
.def    rPocet = r24           // parametr delky pole


//promene ve SRAM
.dseg
pole:   .byte N                                 // promena pole[N]

//program
.cseg
        .org 0X0000
start:  // nacitani cisel z pameti programu
        ldi     ZL, LOW(cisla << 1)             // nastaveni pointru Z na cisla
        ldi     ZH, HIGH(cisla << 1)
        ldi     YL, LOW(pole)                   // nastaveni pointru Y na pozici pole
        ldi     YH, HIGH(pole)
        ldi     rTemp1, N                       // pocitadlo cyklu vyplnovani

plneni: lpm     r0, Z+                          // r1 = Z++
        st      Y+, r0                          // Y++ = r1
        dec     rTemp1                          // rTemp--
        brne    plneni                          // if(rCount2 != 0) skok na plneni

        // priprava volani metody
        ldi     r16, LOW(RAMEND)                // inicializace zasobniku (Stack Pointer)
        out     SPL, R16
        ldi     r16, HIGH(RAMEND)
        out     SPH, R16

        ldi     rPocet, N                       // parametr delka pole
        ldi     YL, LOW(pole)                   // nastaveni pointru Y na pozici pole
        ldi     YH, HIGH(pole)
        rcall   SelShort                        // volani metody SelShort + aktualni PC do zasobniku
```

```
end:    rjmp    end                 // nekonecna smycka

SelShort:       // nastaveni parametru pro spravnou funkci metody
        push    rTemp1              // kvuli uchovani hodnot registru po metode
        push    rTemp2
        push    rPozice
        push    rPozmax
        push    rCount1
        push    rCount2
        push    rCount3
        push    rCislo
        push    rPocet
        push    ZL
        push    ZH

        ldi     ZL, LOW(pole+1)     // nastaveni pointru Z na pozici pole+1
        ldi     ZH, HIGH(pole+1)    // (pomocne pole ve vnorenem cyklu)
        dec     rPocet              // nastaveni pozic countru 1 a 2
        mov     rCount2, rPocet     // rCount2 = rPozice - 1 (pro vnoreny cyklus)
        inc     rPocet
        mov     rCount1, rPocet     // nastaveni rPozice rCount1 (rCount1 = rPozice)
        inc     rPocet
        inc     rPocet              // nastaveni rPozice pro dalsi vypocty
        ld      rTemp1, Y           // pole[0] = rTemp1
zakladni:       // zakladni cyklus
        cpi     rCount2 ,0          // porovnani rCount2 s 0
        breq    konec               // if(rCount2 == 0) skok na konec
        ld      rTemp2, Y           // rTemp2 = pole[Y] (pro porovnani po cyklu)
vnoreny:        // vnoreny cyklus
        ld      rCislo, Z+          // rCislo = pole[Z++]
        inc     rPozice             // rPozice++
        cp      rCislo, rTemp1      // porovnani cisla s rTemp1
        brlt    pokrac              // if(rCislo < rTemp1) skok na pokrac
        mov     rTemp1, rCislo      // rTemp1 = rCislo
        mov     rPozmax, rPozice    // rPozmax = rPozice
        add     rPozmax, rCount3    // rPozmax += rCount3 (spravna rPozice max. prvku)
pokrac:dec      rCount2             // rCount2--
        brne    vnoreny             // if(rCount2 != 0) skok na vnoreny
        // nastaveni pointu Z pro vymenu prvku a nasledna vymena, pripadny skok
        cp      rTemp1,rTemp2       // porovnani rTemp1, rTemp2
        breq    skok                // if(rTemp1==pemp2) skok na skok
        ldi     ZL, LOW(pole)       // nulovani pointru Z (pole[Z] = pole[0])
        ldi     ZH, HIGH(pole)
        add     ZL, rPozmax         // pole[Z] = Z + pole[rPozmax]
        adc     ZH, r1              // ZH += carry (ZH += 0 + carry)
        ld      rTemp2, Y           // rTemp2 = pole[Y]
        st      Z, rTemp2           // pole[Z] = rTemp2
        st      Y, rTemp1           // pole[Y] = rTemp1
        // priprava na dalsi cyklus
```

```
skok:   ldi     ZL, LOW(pole)       // nulovani pointru Z (pole[Z] = pole[0])
        ldi     ZH, HIGH(pole)
        mov     rPozice, rPocet     // rPozice = rPocet
        sub     rPozice, rCount1    // rPozice -= rCount1
        add     ZL, rPozice         // pole[Z] = Z + pole[rPozice]
        adc     ZH, r1              // ZH += carry (ZH += 0 + carry)
        ld      rTemp1, Y+          // nataveni pointru Y rTemp1 = pole[Y++]
        ld      rTemp1, Y           // rTemp1 = pole[Y]
        ldi     rPozmax,0           // rPozmax = 0
        ldi     rPozice,0           // rPozice = 0
        mov     rCount2, rCount1    // vypocet rCount2 (rCount2 = rCount1)
        subi    rCount2, 2          // rCount2 -= 2
        inc     rCount3             // rCount3++
konec:  dec     rCount1             // rCount1--
        brne    zakladni            // if(rCount1 != 0) skok na zakladni

        pop     ZH                  // stav registru pred metodou
        pop     ZL
        pop     rPocet
        pop     rCislo
        pop     rCount3
        pop     rCount2
        pop     rCount1
        pop     rPozmax
        pop     rPozice
        pop     rTemp2
        pop     rTemp1
        ret                         // navrat z metody na pozici PC ze zasovniku

cisla:  .db     2,-15,55,-100,39,6          // ulozena cisla v ROM
```