|  |
| --- |
| Київський національний університет  «Київський політехнічний інститут» |
| Факультет інформатики та обчислювальної техніки |
| Кафедра обчислювальної техніки |
| Лабораторна робота №7 |
| з дисципліни «Системне програмування» |
| *Залікова книжка № 4213* |
|  |
| Виконав студент 3-го курсу |
| групи ІО-42 |
| Кочетов Данило |

|  |
| --- |
| 2016 |

**Завдання**

|  |  |  |
| --- | --- | --- |
| 13 | float b, a[3]; short n,d; b=0;b+=a[n]; | С |

**Лістинг коду**

#include "stdafx.h"

#include "interp.h"

#include "token.h"

#include "visgrp.h"

#include "index.h"

#include "syntaxP.h"

#include "seman.h"

#include "interpF.h"

extern unsigned bcnst32\_buf[MAX\_UCNST];

extern int nInBg, nInCr;

extern struct indStrUS ndxNds[50];

extern struct recrdSMA ftImp;

extern struct recrdSMA ftTbl[170];

extern \_fop \*\_paddf;

int lenth[]={0,0,0,0,1,2,4,8,1,2,4,8,4,8,10};

//#define iMode 0 // первинний режим

#define iMode 0//1 // табличний режим

//#define iMode 2 // режим Асемблера

/\*void\* StIntrp(struct lxNode \* nd, // вказівник на корінь дерева вузлів

int nR) // номер кореневого вузла

{//void \*vp1, \*vp2; // розподіл памяті з кореню дерева, але можна зайти з таблиці

if(nd->ndOp==\_nam||nd->ndOp>=\_EOS)

}\*/

union gnDat acc32;

int cc;

union gnDat stk[32];

unsigned char sPtr=0;

void push(union gnDat d)

{stk[sPtr++]=d;

}

void pop(union gnDat \*pd)

{\*pd=stk[--sPtr];

}

unsigned nLcCr=0;

void clDtLst(struct lxNode \* nd, int tp, unsigned \*pD)

{if(nd->ndOp==\_comma)

{clDtLst(nd->prvNd, tp, pD);

switch(tp)

{case \_f: cnvTo\_f(nd->pstNd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd->pstNd); break;//

case \_ui: case \_si: cnvTo\_i(nd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd->pstNd);

}nLcCr++;//

}else if(nd->ndOp==\_srcn)

{switch(tp)

{case \_f: cnvTo\_f(nd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd); break;//

case \_ui: case \_si: cnvTo\_i(nd->dataType,

(union gnDat \*)(pD+nLcCr),(union gnDat \*)nd->pstNd);

}nLcCr++;//

}

}

struct lxNode \*trmGrdt(struct lxNode \* nd)

{if(nd->ndOp!=\_nam)return trmGrdt(nd->prvNd);

return nd;

}

void prAss(struct lxNode \* nd)

{prLxTxt(nd);

if((nd->prvNd->dataType&0xffe77fff)>=\_f)printf(" => %7.3g ->",acc32.\_fd);

else if(nd->prvNd->dataType&0x00100000)printf(" => %p ->",acc32.\_id);

else printf(" => %7d ->",acc32.\_id);

prLxTxt(nd->prvNd);printf("\n");

}

char dfnFlg=0;

union gnDat\* SmIntrp(struct lxNode \* nd, // вказівник на корінь дерева вузлів

int incR) // кількість повторень

{union gnDat \*vp1, \*vp2;

char\*name; struct lxNode \*nt;

struct indStrUS \*pRtNdx;

if(((nd->ndOp>=\_void&&nd->ndOp<=\_string)

||nd->ndOp>=\_EOS)&&nd->ndOp!=\_remL)

{

if(nd->ndOp>=\_void&&nd->ndOp<\_fork)dfnFlg=1;

if(nd->prvNd&&nd->ndOp==\_ass)

{vp1=SmIntrp(nd->prvNd,1);

if(nd->pstNd->ndOp==\_tdbz)

{nt=trmGrdt(nd->prvNd); nLcCr=0;

clDtLst(nd->pstNd->pstNd,nt->dataType&0x7FF,

(unsigned\*)nt->pstNd);

// if(nd->prvNd->ndOp==\_ixbz)return vp2;

}else{

// vp1=SmIntrp(nd->prvNd,1);

vp2=SmIntrp(nd->pstNd,1);

if((nd->prvNd->dataType&0xffe77fff)>=\_f)

{acc32.\_fd=cnvTo\_f(nd->pstNd->dataType, vp1, vp2);

if(nd->prvNd->dataType>\_f)

acc32.\_dd=cnvTo\_d(nd->pstNd->dataType, vp1, vp2);

}else{if((nd->pstNd->dataType&0xffe77fff)<\_f) acc32.\_id=vp1->\_id=vp2->\_id;

else acc32.\_id=vp1->\_id=vp2->\_fd;}}

prAss(nd);

return &acc32;}

if(nd->prvNd&&nd->ndOp!=\_ixbz&&nd->ndOp!=\_brkz)

{vp1=SmIntrp(nd->prvNd,1);

if((nd->ndOp==\_cln||nd->ndOp==\_else)&&cc!=0)

{if(nd->dataType!=nd->prvNd->dataType)

{if(nd->dataType>=\_f)vp1->\_fd=vp1->\_id;

else vp1->\_id=vp1->\_fd;}

return vp1;}

}

if(nd->pstNd)

{if(nd->ndOp>\_ass&&nd->ndOp<=\_frkz&&nd->prvNd

&&nd->prvNd->ndOp>\_cnst&&nd->pstNd->ndOp>\_cnst)

push(acc32);

vp2=SmIntrp(nd->pstNd,1);

if(nd->ndOp>\_ass&&nd->ndOp<=\_frkz&&nd->prvNd

&&nd->prvNd->ndOp>\_cnst&&nd->pstNd->ndOp>\_cnst)

{--sPtr; vp1=stk+sPtr;}

}

if(nd->ndOp&&nd->ndOp==\_ixbz)

{vp1=SmIntrp(nd->prvNd,vp2->\_id);

// if(nodes[nd->prnNd].pstNd->ndOp==\_tdbz)

}

//вирівнювання типів

if(nd->ndOp>=\_void&&nd->ndOp<\_fork)

{dfnFlg=0;

return NULL;

}

if(dfnFlg!=0||(nd->ndOp>=\_eosP&&nd->ndOp<=\_EOS))

return NULL;

if(//nd->dataType>=\_f ||

(nd->prvNd!=0&&(nd->prvNd->dataType&0x7fff)>=\_f)

||(nd->pstNd!=0&&(nd->pstNd->dataType&0x7fff)>=\_f))

{if(nd->prvNd!=0&&nd->prvNd->dataType<\_f&&nd->ndOp!=\_cln)

vp1->\_fd=vp1->\_id;

if(nd->pstNd!=0&&nd->pstNd->dataType<\_f&&nd->ndOp!=\_ixbz)

vp2->\_fd=vp2->\_id;

switch(nd->ndOp)

{case \_asAdd:

if(nd->prvNd)

{

//printf("asAdd: %.3f %.3f\n", vp1->\_fd, vp2->\_fd);

vp1->\_fd+=vp2->\_fd;

acc32.\_fd=vp1->\_fd;

prAss(nd);}

break;

case \_lt:

if(nd->prvNd)

acc32.\_id=vp1->\_fd<vp2->\_fd;

break;

case \_gt:

if(nd->prvNd)

acc32.\_id=vp1->\_fd>vp2->\_fd;

break;

case \_add:

if(nd->prvNd)

#if iMode==1

{ftImp.oprd1=nd->dataType; ftImp.ln1=nd->resLength;

ftImp.oprd2=nd->dataType; ftImp.ln2=nd->resLength;

ftImp.oprtn=nd->ndOp;

struct recrdSMA\*

pftImp = selBin(&ftImp, ftTbl, 179);

if(pftImp)

acc32=pftImp->pintf(vp1,vp2);/\*\_paddf\*/

}

#else

acc32.\_fd=vp1->\_fd+vp2->\_fd;

#endif

break;

case \_sub:

if(nd->prvNd)

acc32.\_fd=vp1->\_fd-vp2->\_fd; break;

case \_mul:

if(nd->prvNd)

acc32.\_fd=vp1->\_fd\*vp2->\_fd; break;

case \_div:

if(nd->prvNd)

acc32.\_fd=vp1->\_fd/vp2->\_fd; break;

case \_cln:

if(cc==0)

acc32.\_fd=vp2->\_fd; break;

case \_qmrk:

if(nd->prvNd)cc=vp1->\_id;

if(cc)acc32.\_fd=vp2->\_fd; break;

case \_else:

if(cc==0)

acc32.\_fd=vp2->\_fd; break;

case \_if:

if(nd->prvNd)

if(vp1->\_fd) cc=vp1->\_id; break;

case \_ixbz:

if (nd->prvNd) {

//printf("ixbz: %.3f %.3f\n", vp1->\_id, vp2->\_id);

acc32.\_fd = ((float\*)vp1->\_id)[vp2->\_id];

}break;

case \_brkz:

if(nd->prvNd)

acc32.\_fd=vp2->\_fd;

}}else{

switch(nd->ndOp)

{case \_asAdd:

if(nd->prvNd)

{if(nd->prvNd->dataType&cdPtr)

acc32.\_id=vp1->\_id+=vp2->\_id\*lenth[(nd->prvNd->dataType)&0xff];

else acc32.\_id=vp1->\_id+=vp2->\_id;

}

prAss(nd);

break;

case \_asSub:

if(nd->prvNd)

{if(nd->prvNd->dataType&cdPtr)

acc32.\_id=vp1->\_id-=vp2->\_id\*lenth[(nd->prvNd->dataType)&0xff];

else acc32.\_id=vp1->\_id-=vp2->\_id;

}

prAss(nd);

break;

case \_ne:

if(nd->prvNd)

acc32.\_id=vp1->\_id!=vp2->\_id;

break;

case \_addU:

acc32.\_id=vp2->\_id;

break;

case \_add:

if(nd->prvNd)

{if(nd->prvNd->dataType&cdPtr)

acc32.\_id=vp1->\_id+vp2->\_id\*lenth[(nd->prvNd->dataType)&0xff];

else acc32.\_id=vp1->\_id+vp2->\_id;

}

break;

case \_sub:

if(nd->prvNd)

{if(nd->prvNd->dataType&cdPtr)

{if(nd->pstNd->dataType&cdPtr)

acc32.\_id=(vp1->\_id-vp2->\_id)/lenth[(nd->prvNd->dataType)&0xff];

else

acc32.\_id=vp1->\_id-vp2->\_id\*lenth[(nd->prvNd->dataType)&0xff];

}else acc32.\_id=vp1->\_id-vp2->\_id;

}

break;

case \_brkz:

if(nd->prvNd&&nd->prvNd->ndOp!=\_refU)

break;// тутобробка функцій

case \_refU:

acc32.\_id=\*((int\*)(vp2->\_id));

break;

case \_mul:

if(nd->prvNd)

#if iMode==2

\_asm {mov ecx,dword ptr [ebp-4]

mov edx,dword ptr [ebp-8]

mov eax,dword ptr [ecx]

imul eax,dword ptr [edx]

mov dword ptr [acc32],eax

mov dword ptr [acc32+4],edx

}

#else

acc32.\_id=vp1->\_id\*vp2->\_id;

#endif

break;

case \_div:

if(nd->prvNd)

#if iMode==2

\_asm {mov ecx, dword ptr[ebp - 4]

mov edx, dword ptr[ebp - 8]

mov eax, dword ptr[ecx]

idiv eax, dword ptr[edx]

mov dword ptr[acc32], eax

}

#else

acc32.\_id=vp1->\_id/vp2->\_id;

#endif

break;

case \_cln:

if(cc==0)

#if iMode==2

\_asm {

mov edx, dword ptr[ebp - 8]

mov dword ptr[acc32], edx

}

#else

acc32.\_fd=vp2->\_id;

#endif

break;

#

case \_qmrk:

if(nd->prvNd) cc=vp1->\_id;

if(cc)acc32.\_id=vp2->\_id; break;

case \_mod:

if(nd->prvNd)

acc32.\_id=vp1->\_id%vp2->\_id; break;

case \_orB:

if(nd->prvNd)

#if iMode==2

\_asm {mov ecx, dword ptr[ebp - 4]

mov edx, dword ptr[ebp - 8]

mov eax, dword ptr[ecx]

or eax, dword ptr[edx]

mov dword ptr[acc32], eax

}

#else

acc32.\_id=vp1->\_id|vp2->\_id;

#endif

break;

case \_andB:

if(nd->prvNd)

#if iMode==2

\_asm {mov ecx, dword ptr[ebp - 4]

mov edx, dword ptr[ebp - 8]

mov eax, dword ptr[ecx]

and eax, dword ptr[edx]

mov dword ptr[acc32], eax

}

#else

acc32.\_id=vp1->\_id&vp2->\_id;

#endif

break;

case \_xorB:

if(nd->prvNd)

#if iMode==2

\_asm {mov ecx, dword ptr[ebp - 4]

mov edx, dword ptr[ebp - 8]

mov eax, dword ptr[ecx]

xor eax, dword ptr[edx]

mov dword ptr[acc32], eax

}

#else

acc32.\_id=vp1->\_id^vp2->\_id;

#endif

break;

case \_or:

if(nd->prvNd)

acc32.\_id=vp1->\_id||vp2->\_id; break;

case \_and:

if(nd->prvNd)

acc32.\_id=vp1->\_id&&vp2->\_id; break;

case \_ixbz:

if(nd->prvNd)

acc32.\_id=((unsigned\*)vp1->\_id)[vp2->\_id]; break;

case \_dcr:

if(nd->prvNd==0)

{if(nd->pstNd->dataType&cdPtr)

acc32.\_id=(vp2->\_id-=lenth[(nd->pstNd->dataType)&0xff]);

else acc32.\_id=--(vp2->\_id);

}else

{if(nd->prvNd->dataType&cdPtr)

(vp1->\_id=(acc32.\_id=vp1->\_id)-lenth[(nd->prvNd->dataType)&0xff]);

else acc32.\_id=(vp1->\_id)--;

prAss(nd);

}

break;

case \_inr:

if(nd->prvNd==0)

{if(nd->pstNd->dataType&cdPtr)

acc32.\_id=(vp2->\_id+=lenth[(nd->pstNd->dataType)&0xff]);

else acc32.\_id=++(vp2->\_id);

}else {if(nd->prvNd->dataType&cdPtr)

(vp1->\_id=(acc32.\_id=vp1->\_id)+lenth[(nd->prvNd->dataType)&0xff]);

else acc32.\_id=(vp1->\_id)++;

prAss(nd);

}

break;

}}

// pRc=selBin(lPrv, tpTbl, 21);

}else if(nd->ndOp==\_nam)

{if(nd->dataType!=\_v)

{if(nd->pstNd==0)

{pRtNdx=selBTr(nd,ndxNds);// якщо не знайдено - неописане імя

name=(char\*)pRtNdx->pKyStr->prvNd;/\*\*/

nd->pstNd=pRtNdx->pKyStr->pstNd;

if(nd->pstNd==0)

{nd->pstNd=(struct lxNode \*)(bcnst32\_buf+nInCr);

while(incR--)

\*(bcnst32\_buf+nInCr++)=0xCCCCCCCC;

// nInCr+=incR;

}}

}

if((nd->dataType&cdArr)==cdArr)

return vp1=(union gnDat\*)&nd->pstNd;

return vp1=(union gnDat\*)nd->pstNd;

}else if(nd->ndOp==\_srcn)//\_cnst

{return vp1=(union gnDat\*)nd->pstNd;

}else if(nd->ndOp==\_whileP)//\_cnst

{do{

vp1=SmIntrp(nd->prvNd->pstNd,1);

if(vp1->\_id)vp2=SmIntrp(nd->pstNd,1);

else return vp1;

}while(vp1->\_id);

}

return &acc32;

}

**Висновок.**

В ході виконання лабораторної роботи були одержані навички використання вставок на мові Асемблера для побудови та оптимізації абстрактної машини інтерпретації комп’ютерної мови. Вивчені угоди про зв’язки для створення процедур і функцій інтерпретації операцій і операторів комп’ютерних мов і звертання до них за допомогою операторів мови С з використанням функціонального типа даних.