الجلسة الثالثة عشر

توليد الشيفرة الوسيطية

Intermediate Code Generation

الهدف من الجلسة

- تعريف الطالب بكيفية توليد الشيفرة الوسيطية مكتوبة بلغة التجميع (Assembly).

مستلزمات الجلسة

- حاسب بمواصفات دنيا RAM: 1 GB, CPU: 1.6 GHz, Windows 7 OS 32 bit
- Turbo c++
- LEX & BISON tools
- مراجعة لعمليات التحميل والتخزين والإزاحة والعمليات الحسابية والمنطقية للغة التجميع.

خطوات العمل

- توليد الشيفرة الوسيطية (بلغة التجميع)
- كيفية توليد الشيفرة المبدئية
- توليد شيفرة لعمليات التصريح
- توليد الشيفرة الوسيطية لعمليات الإسناد

الخلاصة والنتائج:

يفترض عند نهاية الجلسة:

- تمكن الطالب من تطبيق مثال عملي عن كيفية توليد الشيفرة الوسيطية لعمليات التصريح.

تمكن الطالب من تطبيق مثال عملي عن كيفية توليد الشيفرة الوسيطية لعمليات الإسناد.

1.1 توليد الشيفرة الوسيطية (بلغة التجميع)

بعد أن انتهينا من مراحل بناء المحلل اللفظي والقواعدي والمعنوي بالاعتماد على بيئتي الـ LEX والـ BISON(YACC)، تبدأ مرحلة توليد الشيفرة الوسيطية بلغة التجميع Assembly Language.

سنتعامل مع مجمع assembler 32 bit وسنعتمد على المقاطعات ولن نتعامل مع الدوال الجاهزة وسيكون الكود الناتج أقرب ما يكون إلى لغة الآلة.

1.1.1 كيفية توليد الشيفرة المبدئية:

بداية ننشئ ملفاً باسم CODE_GEN.H وسنكتب فيه بعض الدوال التي ستمكننا من كتابة الكود الوسيط في ملف امتداده .asm

نكتب في هذا الملف التعليمات التالية ونحفظه ونغلق الملف.

أولاً: نكتب تعليمات حجز مقطع المعطيات data_section بحجم 1024 بايت وحجز مقطع البيانات code_section بحجم 8192 بايت في الذاكرة. أما التعليمة الثالثة فتمثل تعريف مقبض للملف الذي سيحمل الكود الناتج هو الملف fcode، كمايلي:

```
#include<stdio.h>
char data_section [1024];
char code_section [8192];
FILE *fcode;
```

بعد ذلك يتم تعريف تابع (دالة) تسمى Init_Code تأخذ بارامتر هو مصفوفة من النوع المحرفي المؤشر اسمها file، يتم في هذا التابع فتح الملف (البارامتر) من أجل عملية الكتابة، ويتم فيه جعل قسم البيانات والكود فارغ كعملية تهيئة أولية.

```
void Init_Code(char *file)
{
fcode = fopen(file, "wb");
strcpy(data_section, "");
strcpy(code_section, "");
}
```

يتم بعد ذلك التصريح عن التابع Add_Data والذي يأخذ كوسيط له البيانات data المراد نسخها إلى قسم البيانات، ثم يقوم بنسخ هذه البيانات باستخدام التابع strcat إلى قسم البيانات.

```
void Add_Data(char *data)
{
strcat(data_section, data);
}
```

يتم بعد ذلك التصريح عن التابع Add_Code والذي يأخذ كوسيط له الشيفرة code المراد نسخها إلى قسم التعليمات، ثم يقوم بنسخ هذه البيانات باستخدام التابع strcat إلى قسم التعليمات.

```
void Add_Code(char *code)
{
```

```
strcat(code_section, code);
}
```

يتم هنا التصريح عن تابع الإخراج Dispose_Code الذي سيقوم بطباعة التوجيهات التي ستظهر في ملف الخرج .fcode

يمكن مشاهدة ناتج هذه التعليمات على ملف الخرج

```
void Dispose_Code()
{
fprintf(fcode, ".MODEL small\r\n.stack 100h\r\n.DATA\r\n");
fprintf(fcode, "%s", data_section);
fprintf(fcode, ".CODE\r\nstart:\r\n");
fprintf(fcode, "mov ax,@data\r\n");
fprintf(fcode, "mov ds,ax\r\n");
fprintf(fcode, "%s", code_section);
fprintf(fcode, "end start\r\n");
fclose(fcode);
}
```

نكتب التعليمات السابقة ضمن ملف (CODE_GEN.H) حيث لا بد من تضمين هذا الملف ضمن ملف المعرب ضمن قسم التضمين كمايلي:

```
#include"C:\Lex_Yacc\examples\code\CODE_GEN.H"
```

كذلك لابد من تهيئة الملف الذي سنكتب فيه الكود المولد وهذا سيكون ضمن التابع الرئيس main() كمايلي:

```
Init_Code("C:\\lex_yacc\\exemples\\test.asm");
yyparse();
Dispose_Code();
```

وعند تنفيذ الملفات من جديد لبناء المترجم مع استخدام ملف دخل input.txt خال من الأخطاء، سنحصل على ملف هو الملف test.asm إذا فتحناه سنشاهد فيه توجيهات التهيئة الافتراضية التي قمنا بطباعتها من خلال استدعاء الدالة Init_Code التي قامت بتهيئة الملف test.asm ثم تم استدعاء الدالة Dispose_Code(); التي قامت بطباعة توجيهات التهيئة إلى ملف الخرج test.asm. عند فتح ملف test.asm سنجد فيه مايلي:

ملف الشيفرة المبدئي

هنا الملف فارغ ولا يحتوي أي تعليمات أو بيانات والمراحل التالية ستبين كيفية إضافة البيانات لجزء البيانات والتعليمات لجزء التعليمات.

1.1.2 توليد الشيفرة المقابلة لعمليات التصريح:

سنقوم كخطوة أولى بإضافة البيانات إلى قسم البيانات data في ملف لغة التجميع. كل تعليمة من تعليمات التصريح الموجودة في الملف المصدري لها تعليمات مقابلة في لغة التجميع، والأمثلة التالية توضح ذلك:

real x; ---------→ dd 6 dup(?)

int g; ---------→ dd 6 dup(?)

char c ; ---------→ db (?)

chain k; ---------→ db 50 dup(?)

تعني هذه التعليمات بالترتيب مايلي:

تعريف أربع بايتات في قسم المعطيات ليس لها قيمة مبدئياً (تمثل التصريح عن متغير من النوع الصحيح باللغة عالية المستوى).

تعريف أربع بايتات في قسم المعطيات ليس لها قيمة مبدئياً (تمثل التصريح عن أربعة بايتات من النوع real باللغة عالية المستوى).

تعريف بايت واحد ليس له قيمة مبدئياً (تمثل التصريح عن متغير من النوع char).

تعريف 50 بايت ليس لها قيمة مبدئياً (تمثل التصريح عن متغير من النوع chain أو سلسلة).

والسؤال المطروح الآن: كيف يمكن إضافة هذه التعليمات المكتوبة بلغة التجميع إلى الملف test.asm الذي تم إنشاؤه؟

الجواب: باستخدام التابع Add_Data الذي قمنا بتعريفه على مستوى الملف CODE_GEN.H.

والآن نعدل ملف وصف المعرب كمايلي:

في كل قاعدة تتضمن عملية تصريح يجب أن تتم إضافة بيانات إلى قسم البيانات من ملف الآسكي.

لدينا قاعدتان للتصريح عن المتغيرات هما:

ident:ID|

ID ident

نعدل هاتين القاعدتين كمايلي:

```
ident:ID {setup_sym($1,current_type);
if((current_type == 1) ||
(current_type == 2))
```

```
{
Add_Data($1);
Add_Data("\tdd\t6 dup(?)\r\n");
}
else if(current_type == 3)
{
Add_Data($1);
Add_Data("\tdb\t?\r\n");
}
else if(current_type == 4)
{
Add_Data($1);
Add_Data("\tdb\t50 dup(‘ ‘)\r\n");
}
}
|ID ident {setup_sym($1,current_type);
if((current_type == 1) ||
(current_type == 2))
{
Add_Data($1);
Add_Data("\tdd\t6 dup(?)\r\n");
}
else if(current_type == 3)
{
Add_Data($1);
Add_Data("\tdb\t?\r\n");
}
else if(current_type == 4)
{
Add_Data($1);
```

```
Add_Data("\tdb\t50 dup(‘ ‘)\r\n");
}
};
```

يتم في التعليمات المضافة إلى ملف الماسح التحقق من أن المتغير هو من النمط الصحيح أو النمط ذي الفاصلة العائمة، فإذا تحقق ذلك فإنه تتم إضافة المعطيات التالية لقسم المعطيات:

```
Add_Data($1);
Add_Data("\tdd\t6 dup(?)\r\n");
```

أي قم بإضافة المتغير ID المعرف بإشارة $1 إلى قسم المعطيات data، ثم قم بترك فراغ بعده قم بوضع العبارة dd ثم يتم وضع فراغ \t ثم الرقم 6 ثم توضع العبارة dup ثم إشارة (?) ثم سطر جديد \n، وهذا سيضع العبارة التالية في قسم المعطيات كمايلي:

```
ID-name dd 6 dup(?)
```

وهذه العبارة تعني التصريح عن متغير اسمه x حجمه بمقدار ست بايتات. أما الرقم 6 فيعني أن هذا الرقم الصحيح أو ذو الفاصلة العائمة الذي تم تعريفه سيكون مؤلف من 6 خانات كحد أعظمي، وهذا هو مبدأ الأعداد الصحيحة التي تحتاج لأربع بايتات في الذاكرة كحجم لتشغله (dd) وهذا يعني أن العدد الأعظمي لخانات الرقم الصحيح هو 6.

أما إن كان نمط البيانات هو من النوع المحرفي char فستتم إضافة المعطيات التالية:

```
Add_Data($1);
Add_Data("\tdb\t?\r\n");
```

أي قم بإضافة المتغير ID المعرف بإشارة $1 إلى قسم المعطيات data، ثم قم بترك فراغ وبعده قم بوضع العبارة db ثم فراغ ثم إشارة ؟، وهذا يعني أننا سنحصل على العبارة التالية في قسم المعطيات:

```
ID-name db ?
```

أي التصريح عن متغير من النوع char وقيمته غير معروفة حالياً.

وإذا كان النمط هو السلسة string ستتم إضافة المعطيات التالية:

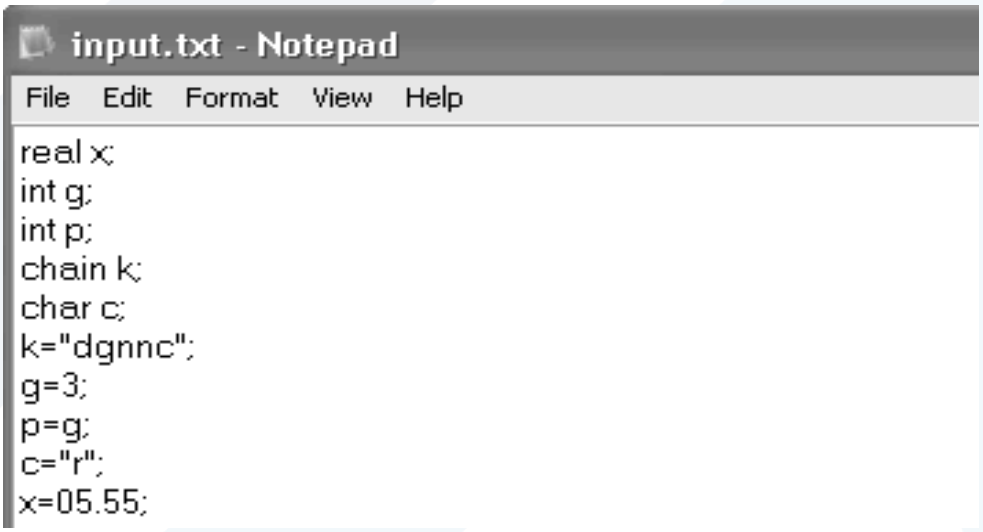
```
Add_Data($1);
Add_Data("\tdb\t50 dup(‘ ‘)\r\n");
```

أي قم بإضافة المتغير ID المعرف بإشارة $1 إلى قسم المعطيات data، ثم قم بترك فراغ وبعده قم بوضع العبارة db ثم فراغ ثم الرقم 50 ثم إشارة ؟، وهذا يعني أننا سنحصل على العبارة التالية في قسم المعطيات:

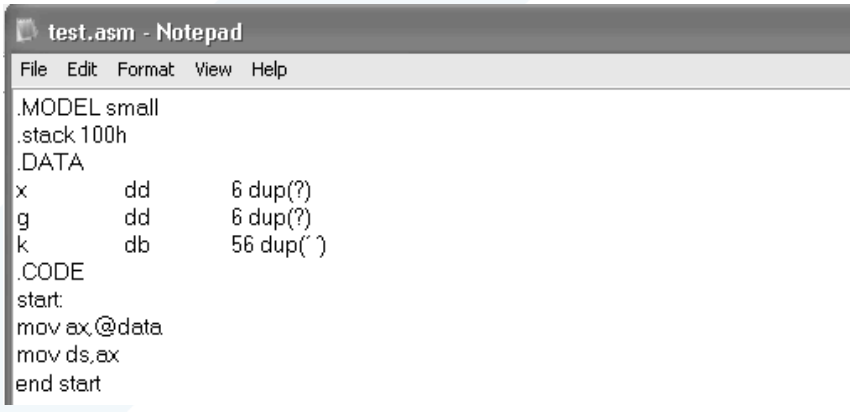
ID-name db 50 dup(‘ ‘)

أي التصريح عن سلسلة بحجم أعظمي 50 محرف وهذه السلسلة فارغة حالياً.

والآن نحفظ التغييرات التي قمنا بإجرائها ونعيد عملية الترجمة للحصول على الملف test.asm وعند فتحه سنشاهد فيه ماتم شرحه في الأعلى كما يوضح الشكل الآتي:



ملف المدخلات المصدري



توليد الشيفرة الوسيطية لعمليات التصريح

والسؤال الآن من أين حصلنا على أسماء المتغيرات k , g , x والجواب: من ملف المدخلات input.txt والذي يحتوي على ثلاث عمليات إسناد وثلاث عمليات تصريح، مايهمنا حتى هذه النقطة هو عمليات الإسناد فقط وهي:

```
real x;

int g;

chain k;
```

انتهت الجلسة – د. علي ميا ، م. رشا شباني