CS340400 Compiler Design Homework 3

Demo Time

2018/06/19, 20, 21

NTHU PL LAB

Submission

- **You must upload all 2 items: your source code (lex scanner and yacc parser+code generator), a makefile in server, or you will get zero credit!**
- Server: Source code
  - **Must** create “hw3” under your home directory
    - e.g. Student ID = 104062634 Your home directory is /home/104062634/hw3
  - In your home directory/hw3, you **must** provide
    - The revised version of your source code.
    - A **makefile** for you to compile your code.
  - The makefile in which the name of the output executable file **must** be named ‘**codegen**’ .
  - If you include other files in your source code, remember to upload them, too!

Demo Time

- **TBA**
  - **6/19(Tue.)~6/21(Thur.)**

關於大四畢業生成績

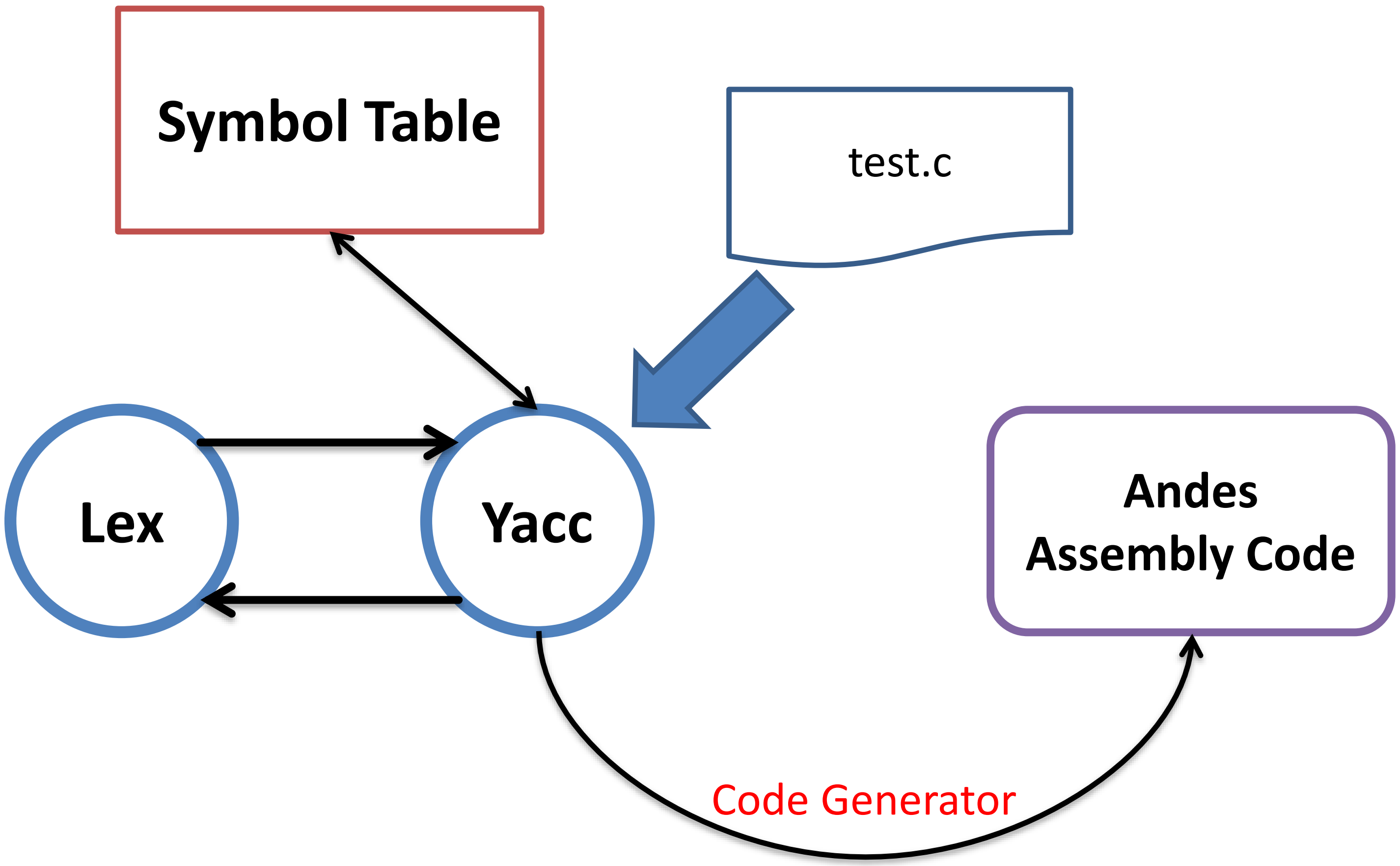
- 假設全部畢業生必須於最晚6/25(一)之前送交成績，請各位畢業生於6/13(三)~6/15(五)，中午時間前往指定地點demo HW3作業。
- 有來Demo，就優先批改期末考卷。

- Demo地點: 綜二R734
- Demo時間: PM12:00~02:00

HW3\ Directory

- HW3\
  - HW3 Environment\
    - HW3_板子環境建置
  - ProfLee_HW3_Template\
    - 範本(只實作+, *)
  - Testcase1\
    - 65分case(四則運算，補減法和除法)
  - Tools and References\
    - 建置環境及編譯所需工具
    - Andes ISA參考資料

HW3 Flow


Symbol Table
test.c
Lex
Yacc
Andes Assembly Code
Code Generator

Andino Code

- Use $sp instead of $fp
- You do not need to deal with prologue/epilogue (push.s {}, pop.s {})

Andino Code (Cont.)

Registers

fp: Frame pointer.

sp: Stack pointer.

lp: Return address of caller.

r0: Store return value.

r0 ~ r5: Pass argument.

```
// main function
int main() {
  int a = 3000;
  int b = 3000;

  b = a / 3 + b * 3 - b;
  return 0;
}
```

Three Parts of Implementation

- Add symbol table
- Add assembly code
- Generate assembly file

Add Symbol Table

- A table which keeps the information of symbol.
  - E.g. scope, type, parameters offset…
- Implement in other C files.
- When you scan a variable, you should store the information of variable into symbol table.

```
struct symbol {
    char      name[32];
    double  value;
    int         offset;
    …….
};
```

index	name	value	offset	……
0	a	2000	4	……
1	b	35	8	……
2	c	-23	12	……
3	d	7.5	16	……

⋮

Add Assembly Code

- YACC

```
%{
#include "symbol.h"
%}

%union {
   int intVal;
   struct symbol *sym;
}

%%
…….

expr: VAR '+' NUM {
     $$.intVal = $1.sym->value + $3.intVal;
     fprintf(f_asm," movi $r0, %d\n",$1);
     fprintf(f_asm," movi $r1, %d\n",$3);
     fprintf(f_asm," add $r0, $r0, $r1\n");

};
```

…….

Generate Assembly File

- Generate Assembly File

```
Declaration:  FILE * f_asm;
f_asm is a file descriptor.

if(  (f_asm = fopen("assembly", "w")) == NULL ) {
     fprintf(stderr, "Can not open the file %s for writing.\n", "assembly");
}
```

```
fprintf(f_asm, "movi   $r0, %d", $1.intVal);
```

```
fclose(f_asm);
```

How to execute assembly of HW3

準備工作

- 自己寫的程式

- 助教提供的檔案

  HW3\Tools and References\Assembly_Combiner
    - create.sh : 製作Blink.s的shell script
    - upper.s : Andes組合語言制式開頭部分
    - lower.s : Andes組合語言制式結尾部分

  HW3\Tools and References\DemoBlink
    - Cygwin環境下編譯程式所需要的資料

Step1: Windows環境設定

- 請參照 HWЗ\HW3 Environment\
  1_Andes Andino Environment Guide.pdf
  2_HW3 Environment Guide.pdf

  將執行環境建立起來

Step2: 作業執行流程

- **請使用自己開發好的編譯器將HW3\Testcase1\test.c編譯成組合語言檔，名稱為 “assembly”**
  - **此組合語言檔不需要處理 “ int main() “和 “ return “，只需保留程式主體的組合語言部分**
- **用助教提供的**
  HW3\Tools and References\Assembly_Combiner\
  裡面的create.sh, 在Server或Cygwin環境下將assembly, upper.s, lower.s 這三個檔案組合產生Blink.s
- **最後，將Blink.s放到Cygwin環境下的正確位置，編譯成執行檔，透過燒錄動作放入到板子執行**

如何觀察Andes組合語言

- HW3\Testcase1\
  - test.c
  - Blink.cpp

```
// test.c
int main() {
  int a = 3000;
  int b = 3000;

  b = a / 3 + b * 3 - b;
  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
  return 0;
}
```

```
// Blink.cpp
#include <arduino.h>

void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  int a = 3000;
  int b = 3000;

  b = a / 3 + b * 3 - b;

  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
}
```

- 將Blink.cpp放置到Cygwin環境下的 DemoBlink/Demo_Blink底下
  - 注意: Demo_Blink目錄內只能存在Blink.cpp 或 Blink.s

如何觀察Andes組合語言(Cont.)

- 開啟 Cygwin terminal，預設執行目錄為 DemoBlink/
  - $ make
  - $ cd Demo_Blink
  - 產生的檔案中，會有Blink.o.s，透過一般的編輯器打開來觀看Andes gcc轉出來的組合語言

- 以上是以HW3\Testcase1 為範例，其他程式內容，依上述類推來觀察

upper.s? lower.s?

- upper.s
- lower.s

```
// Blink.cpp
#include <arduino.h>

void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  int a = 3000;
  int b = 3000;

  b = a / 3 + b * 3 - b;

  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
}
```

Exception Handling

- 紅框處，HW1和HW2沒有要求，所以這邊請自行從Lex到yacc，特別處理digitalWrite, delay, LOW, HIGH等的字串，例外判斷並輸出如下示範:

```
movi      $r0, 13
movi      $r1, 1      ← high
bal       digitalWrite
lwi       $r0, [要放入的值]
bal       delay
movi      $r0, 13
movi      $r1, 0      ← low
bal       digitalWrite
lwi       $r0, [要放入的值]
bal       delay
```

```
// test.c
int main() {
  int a = 3000;
  int b = 3000;

  b = a / 3 + b * 3 - b;
  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
}
```

Exception Handling(Cont.)

- 如果選擇Blink.cpp為輸入檔，並直接產生完整Andes組合語言。請自行從Lex到yacc，特別處理相關字串，例外判斷並輸出。
  - " #include <Arduino.h> "
    - 需例外處理避開
  - Output 代表 1
  - High 代表 1
  - Low 代表 0

```
// Blink.cpp
#include <arduino.h>

void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  int a = 3000;
  int b = 3000;

  b = a / 3 + b * 3 - b;

  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
}
```

How to get your grade

D	= 50
C	= 65
B	= 80
A	= 100

Notice

- Grade C, B這三種等級所使用的公開正常測資(testcase1/test.c 及page 27, 37)及隱藏測資，一律只使用int為變數宣告型態
- HW3所有spec及測資語法規範，均繼承HW1和HW2

Grading Policies - Grade D

Requirements:

- Students part
  - Source code can not compile or execute
  - No need any report
  - Must participate the demo, **or you will get zero**
- TA part
  - Ask some question about your source code of HW3

Grading Policies - Grade C

Requirements:

- Students part
  - Pass HW3\Testcase1\test.c
  - Burn your binary file into Andino board and show the result
  - No need any report
  - Must participate the demo, **or you will get zero**
- TA part
  - Use 1 or 2 case to verify your demo result
  - Ask some question about your source code of HW3

Grading Policies - Grade B

Requirements:

- Students part
  - Pass HW3\Testcase1\test.c
  - Add statement of if, if else, and while loop
  - Burn your binary file into Andino board and show the result
  - No need any report
  - Must participate the demo, **or you will get zero**
- TA part
  - Use 1 or 2 case to verify your demo result
  - Ask some question about your source code of HW3

Grade B以下列三種為公開測資

```
// if case, just example
int main() {
   int a = 2000;

   digitalWrite(13, HIGH);
   delay(a);
   digitalWrite(13, LOW);
   delay(a);

  if (a != 0)
  {
    int b = 4000;
    int c = 2000;
    a = (b * 2 + c)/2;
    digitalWrite(13, HIGH);
    delay(a);
    digitalWrite(13, LOW);
    delay(a);
  }
  return 0;
}
```

```
// if else case, just example
int main() {
   int a = 2000;

  if (!a)
  {
    int b = 4000;
    int c = 2000;
    a = (b * 2 + c)/2;
    digitalWrite(13, HIGH);
    delay(a);
    digitalWrite(13, LOW);
    delay(a);
  }
  else
  {
    digitalWrite(13, HIGH);
    delay(a);
    digitalWrite(13, LOW);
    delay(a);
  }
   return 0;
}
```

```
// while case, just example
int main() {
   int a = 1;

   while(a < 10)
   {
     int b = 0;
     b = a * 1000;
     digitalWrite(13, HIGH);
     delay(b);
     digitalWrite(13, LOW);
     delay(b);
     a=a+1;
   }
   return 0;
}
```

沒有nested!!!

TA只會一個測資出現其中一種，不會混合。每種敘述只會出現一次，意思是不會出現兩個if或while。

Grading Policies - Grade A

Requirements:

- Students part
  - Pass HW3\Testcase1\test.c
  - Add statement of if, if else, and while loop
  - Error handling
    - Just print out with the format below:
      Error at line [linenum]: [error message]
  - Burn your binary file into Andino board and show the result
  - No need any report
  - Must participate the demo, **or you will get zero**
- TA part
  - Use 1 or 2 group cases to verify your demo result
  - Ask some question about your source code of HW3

REQUIREMENTS: ERROR HANDLING

Types

- **void**
- Scalar type:
  - **int**, **double**, **char, and bool**
- You don’t need to handle arrays!
- No pointers in our homework!

Expressions(*Expr*)

- Only need to handle the following operators: * + - /
  - Only need to take care of A*B, A+B, A-B, A/B where A and B are of the same types.
  - The operands' types must be either integer or double.
- Only need to handle the following legal components:
  - literal constants
  - a single identifier
  - function invocations with the form: Ident(0 or multiple *Expr* separated by commas)
    - E.g. foo()

Function

- A function’s declaration must appear before its definition, and the definition must match its declaration. (**Except main,** once a function is declared, it must be defined somewhere after the declaration.)
- A function can be declared or defined only once.
- A function must be declared or defined before it is invoked.
- The variable returns by a return statement must match the return type of the function’s declaration or definition.
  - A return statement can only be used inside an non-void function
  - For an non-void function, the last statement of the function’s definition must be a return statement.
- The types of the arguments must be identical to the parameters in the function’s declaration and the function’s definition.

Variables

- Can not use a variable that is not declared.
- A variable can be declared only once.
- For variable initializations, the type of the left-hand side must be the same as the type of the right-hand side.

Compound Statements

```
{
        0 or multiple variable and constant declarations
        0 or more statements
}
```

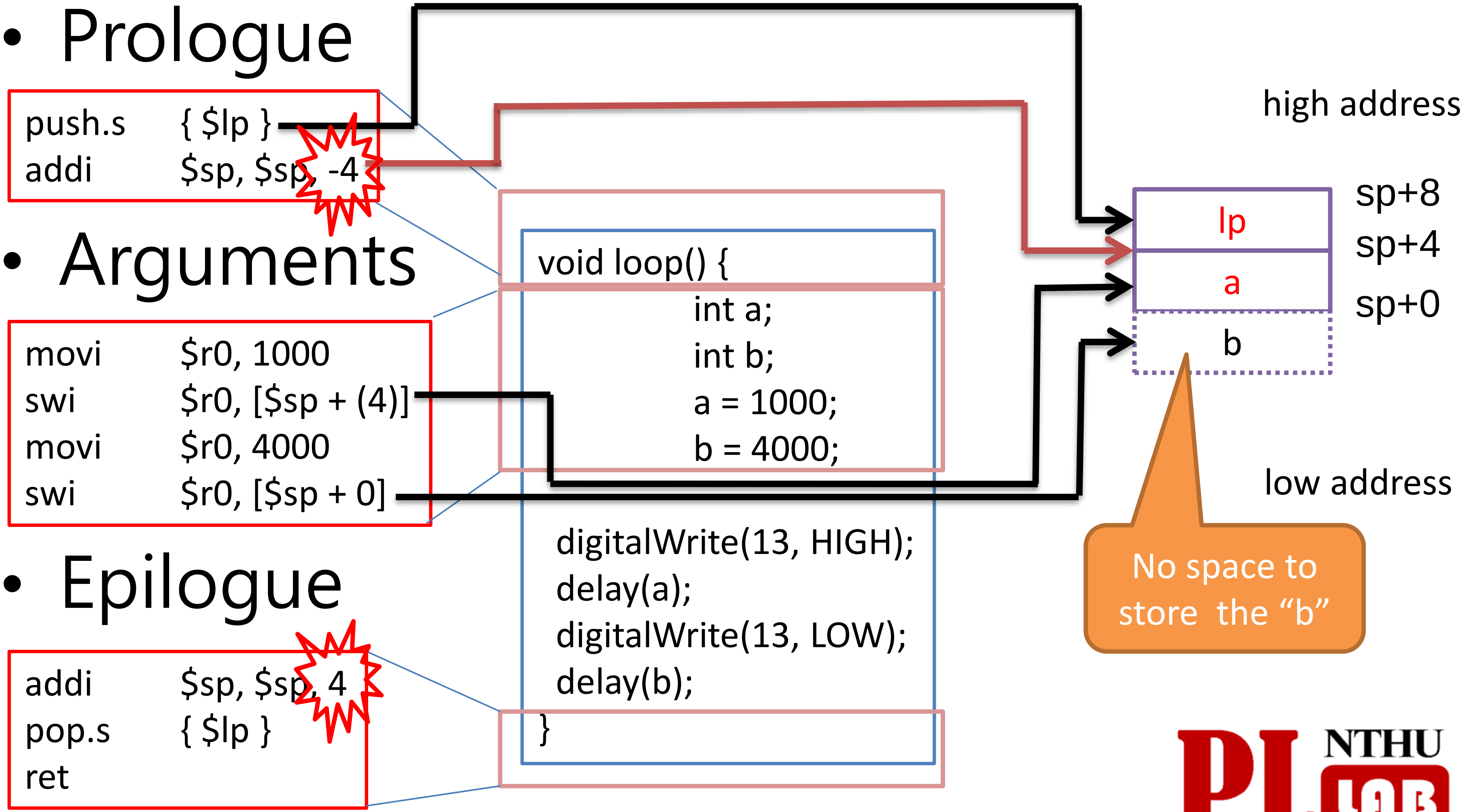
- A compound statement forms an inner scope.
- A variable declared in a compound statement is accessible in the block and all inner blocks of that compound statement, but not accessible outside the compound statement.

break, continue Statements

- break statements can only appear in switch and loop statements.
- continue statements can only appear in loop statements.

Appendix

Stack Overflow Issue



- Prologue

```
push.s      { $lp }
addi        $sp, $sp, -4
```

- Arguments

```
movi        $r0, 1000
swi         $r0, [$sp + (4)]
movi        $r0, 4000
swi         $r0, [$sp + 0]
```

- Epilogue

```
addi        $sp, $sp, 4
pop.s       { $lp }
ret
```

```
void loop() {
        int a;
        int b;
        a = 1000;
        b = 4000;

  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
}
```

Fixed Stack Overflow

- Prologue

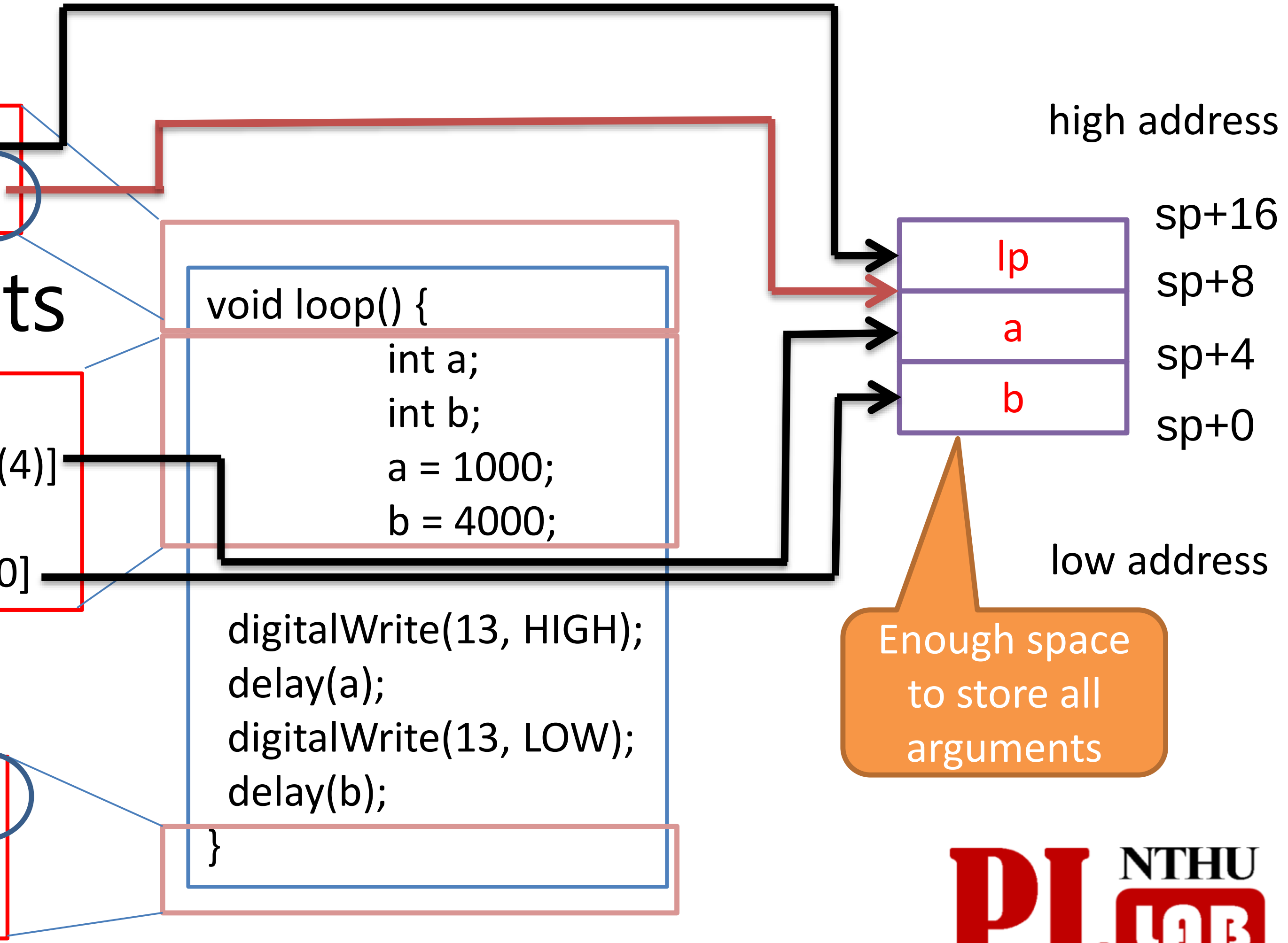
```
push.s    { $lp }
addi      $sp, $sp, -8
```

- Arguments

```
movi      $r0, 1000
swi       $r0, [$sp + (4)]
movi      $r0, 4000
swi       $r0, [$sp + 0]
```

- Epilogue

```
addi      $sp, $sp, 8
pop.s     { $lp }
ret
```



```
void loop() {
            int a;
            int b;
            a = 1000;
            b = 4000;
  digitalWrite(13, HIGH);
  delay(a);
  digitalWrite(13, LOW);
  delay(b);
}
```

Adjust upper.s and lower.s

- HW3\Tools and References\Assembly_Combiner\
  upper.s default setting:

```
push.s      { $lp }
addi        $sp, $sp, -12
```

  lower.s default setting:

```
addi        $sp, $sp, 12
pop.s       { $lp }
ret
```

The number 12, it means that we can store 3 arguments. If you want to store 4 arguments, please manually adjust upper.s and lower.s.

End