CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2013
Assignment #12
30 Points + 15 points extra credit
Due: November 26, 2013

Part 1: Reading

Compilers - Principles, Techniques, & Tools, 2nd Edition, Sections 8.5 to 8.9, 9.1 to 9.2.
Homework Exercises – laboratory assignment and answer the questions at the end of the assignment.

Introduction

In the last assignment you optimize the code generation by improving QUAD generation and/or optimized the QUAD linked list. In this assignment you modify the calculator program to generate PIC 16F1827 assembler code. You support QUAD to TUPLE generation for expressions and IF-ELSE statements and add support for indirect array quads. You test your code generator with the previous test files. You update the compiler and implement the

For extra credit (10) you update the code generator or write your own version. For additional extra credit (5) you download MPLAB IDE software, compile and simulate your output code.

Part 2: Laboratory

From a console window, make a directory on your computer in your EECS337 directory under your Case ID and call it hw12.
mkdir ~/EECS337/caseid/hw12/ ; where caseid is YOUR Case ID, enter all in lower case

Change directory to the hw12 directory.
cd ~/EECS337/caseid/hw12/

Download a copy of: hw12_caseid.tar file to the hw12 directory from http://blackboard.case.edu/ in the EECS337 homework assignment area. To untar the tar file type the command:
tar xvf hw12_caseid.tar

The following files will be created in the current working directory.
codegen.c
codegen2.c
Makefile
hw12_test.sh

Copy the following files from the assignment 10 directory to this directory with the commands:
cp ../hw11/lex.l .
cp ../hw11/main.c .
cp ../hw11/quad.c .
cp ../hw11/symbol_table.c .
cp ../hw11/yacc.y .
cp ../hw11/yystype.h .

Optional: If you did not complete the last assignment you can download the hw11solutions.tar file from the assignment directory and use that as your starting code. Use the command "tar xvf hw11solutions.tar". The hw11/ directory will be created with the files. In that case edit all the files and change "caseid" to your Case ID.

You are now ready to solve the laboratory assignment.

Part 3: Laboratory Assignment
Edit the yystype.h file and add to the top of the file the PIC instructions and the TUPLE structure definition.

```
/*
 *     define pic instructions
 */
#define I_LABEL            0
#define I_MOV       1
#define I_ADD       2
#define I_AND       3
#define I_IOR       4
#define I_SUB       5
#define I_XOR       6
#define I_COMF             7
#define I_DECF             8
#define I_DECFSZ    9
#define I_INCF             10
#define I_INCFSZ    11
#define I_RLF       12
#define I_RRF       13
#define       I_SWAPF             14
#define I_BCF       15
#define I_BSF       16
#define I_BTFSC            17
#define I_BTFSS            18
#define I_CALL             19
#define I_GOTO             20
#define I_TRIS             21
#define I_CLR       22
#define I_RETLW            23
#define I_CLRWDT    24
#define I_NOP       25
#define I_OPTION    26
#define I_RETFIE    27
#define I_RETURN    28
#define I_SLEEP            29
/*
 *     define a tuple structure
 *     supports: CONSTANT STRING_LITERAL IDENTIFIER types
 */
#define       TUPLE  struct tuple
TUPLE
{
       TUPLE         *next;
       int           token;
       unsigned char value;
       int           address;
#define       MASK_VALUE    0x0001
#define       MASK_ADDRESS  0x0002
#define       MASK_LABEL    0x0004
#define       MASK_W_REG    0x0008
#define       MASK_F_REG    0x0010
#define       MASK_INSTR    0x0020
       int           mask;
       char          *buffer;
       int           length;
       int           level;
};
```

Add to the bottom of the symbol table data structure the field to hold the PIC physical address.

```
       int    address;
```

Add to the bottom of the data structure the field to hold the PIC physical address and define the values for the maximum and minimum PIC 16F1827 address ranges.

```
#define       TOP_MEMORY    0x0020
#define       BOTTOM_MEMORY 0x007f
       int    address;
```

Before the bottom of the file (#endif) add the external declarations below for the code generator.

```
/*
 *     external variables and functions from codegen.c
 */
extern void   code_generator_pic_prefix( void);
extern void   code_generator_pic_postfix( void);
extern void   code_generator_operand_postfix( TUPLE *tuple);
extern void   code_generator_operand( TUPLE *tuple);
extern void   code_generator_instr_postfix( TUPLE *tuple);
extern void   code_generator_instr( TUPLE *tuple);
extern void   code_generator_pic16f1827( TUPLE *tuple_list);
extern void   code_generator_instr_test( void);
/*
 *     external variables and functions from codegen2.c
 */
extern void   code_generator_pic_address( void);
extern TUPLE  *generate_quad_instruction( int instruction, int type, int index);
extern TUPLE  *generate_quad_operand( int type, int index);
extern TUPLE  *generate_quad_destination( int type, int index);
extern TUPLE  *generate_quad( QUAD *quad);
extern TUPLE  *generate_quad_2_tuple( QUAD *quad_list);
```

Save the yystype.h file

Edit the main.c and add to the '+' flags section the code below. This flag prints out examples of the PIC code generator instructions.

```
              else if( !strncmp( command, "+test", strlen( command)))
              {
                     code_generator_instr_test();
                     exit( 0);
              }
```

Save the main.c file.

Edit the symbol_table.c file and add the code to allocate the PIC physical address. In the new_symbol function add one line of code to each CHAR, SHORT, INT and LONG specifiers.

```
/*
 *     allocate the PIC physical address
 */
            data.st[ identifier].address = get_address( 1); //char
            data.st[ identifier].address = get_address( 2); //short
            data.st[ identifier].address = get_address( 4); // int
            data.st[ identifier].address = get_address( 8); //long
```

Save the symbol_table.c file

Edit the lex.l file and delete the line that has [\n] and add the new line character to the white space regular expression [\n\t]. From now on we want to generate the whole file or until we type $ on the command line. Save the lex.l file.

```
[ \t\n]                    { /* ignore white characters */ }
```

Edit the yacc.y file and at the top change the include files, the global TUPLE variable and the new %start file for the start symbol.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <ctype.h>
#include "yystype.h"
#include "y.tab.h"
#include "codegen.c"
#include "codegen2.c"
TUPLE   *tuple_list;
%}

%start file
```

Change the top-level productions to use the productions shown below.

```
file   : lines
       {
/*
 *     print the quad list
 */
              print_quad_list( $1.quad );
/*
 *     generate the tuple from quad list
 */
              tuple_list = generate_quad_2_tuple( $1.quad);
/*
 *     free the quad list
 */
              free_quad_list( $1.quad);
/*
 *     call the code generator
 */
              code_generator_pic16f1827( tuple_list);
       }
       ;
lines  : lines stmts
       {
              $$.quad = end_quad_list( $1.quad);
              if( $$.quad)
              {
                  $$.quad->next = $2.quad;
                  $$.quad = $1.quad;
              }
              else
                  $$.quad = $2.quad;
       }
       | lines decls
       {
              $$.quad = end_quad_list( $1.quad);
              if( $$.quad)
              {
                  $$.quad->next = $2.quad;
                  $$.quad = $1.quad;
              }
              else
                  $$.quad = $2.quad;
       }
       | stmts
       | decls
       ;
```

```
decls  : type ident ';'
       {
             new_symbol( $1.index, $2.index, 0);
             $$.quad = (QUAD*)0;
       }
       | type ident '[' number ']' ';'
       {
             new_symbol( $1.index, $2.index, $4.index);
             $$.quad = (QUAD*)0;
       }
       ;
```

Save the yacc.y file.

Build the calc program and fix any errors using the commands: **make clean** and **make**

To test your version, type:
./calc +symbol and enter the lines below.

```
for caseid start time: Mon Nov 18 10:45:08 2013
Enter calculator expression and $ to exit
char a;
a = 10;
$
       a = 10
; automatic code generation for PIC16F1827
; EECS337 Compiler Design
; by: caseid, date: Fall 2013
; for PIC16F1827 processor
; CPU configuration
       list          p=16f1827      ; list directive to define processor
       #include      <p16f1827.inc> ; processor specific variable definitions
       __CONFIG _CONFIG1, _FOSC_INTOSC & _WDTE_OFF & _PWRTE_OFF & _MCLRE_ON &
_CP_OFF & _CPD_OFF & _BOREN_OFF & _CLKOUTEN_ON & _IESO_OFF & _FCMEN_OFF
       __CONFIG _CONFIG2, _WRT_OFF & _PLLEN_OFF & _STVREN_OFF & _BORV_19 &
_LVP_OFF
; generate symbol table:
a_ EQU 0x20
; generate temporary addresses:
; power-up or reset vector
       org     0x0000            ; processor reset vector
       pagesel init
       goto   init   ; skip interrupt vector space (reserved)
; interrupt vector
       org    0x04
inter:              ; on interrupt PC set to 0x04 and automatically clears GIE
       retfie       ; return from interrupt and automatically sets GIE
; beginning of program code
       org    0x08
init:
; On reset all ports are inputs.
       movlb  1      ; switch to bank 1 memory
;      clrf   TRISA         ; set PORTA to all outputs
       clrf   TRISB         ; set PORTB to all outputs
       movlb  0      ; switch to bank 0 memory
mloop:
; read from the standard input (stdin)
       movf   PORTA,w
       nop
       movlw  0xa
       movwf  a_
       goto   mloop
```

```
; only standard library function
printf:
        movwf  PORTB  ; output w to standard output (stdout)
        return
        end           ; end of program code
symbol table:
index: 1 identifier: a length: 2 specifier: char
index: 2 constant: 10 length: 3 format: decimal
```

Test using the test files using the commands below. Determine what is missing from the code generation and expand the compiler to generate the correct output.
./calc +symbol ../hw09/math21.txt

Part 4: Output Generation
When all your lab assignments have been completed execute the homework script file "./hw12_test.sh" using the command below.
./hw12_test.sh &> hw12_test.txt

Print out the hw12_test.txt file and put your name, assignment number and date on it. Turn in the file for the assignment and answer the questions at the end of the assignment.

Your final directory structure for the calc compiler should be as below (using your Case ID):
EECS337/caseid/hw12/Makefile
EECS337/caseid/hw12/calc
EECS337/caseid/hw12/codegen.c
EECS337/caseid/hw12/codegen2.c
EECS337/caseid/hw12/hw12_caseid.tar
EECS337/caseid/hw12/hw12_test.sh
EECS337/caseid/hw12/hw12_test.txt
EECS337/caseid/hw12/lex.l
EECS337/caseid/hw12/main.c
EECS337/caseid/hw12/quad.c
EECS337/caseid/hw12/symbol_table.c
EECS337/caseid/hw12/yacc.y
EECS337/caseid/hw12/yystype.h

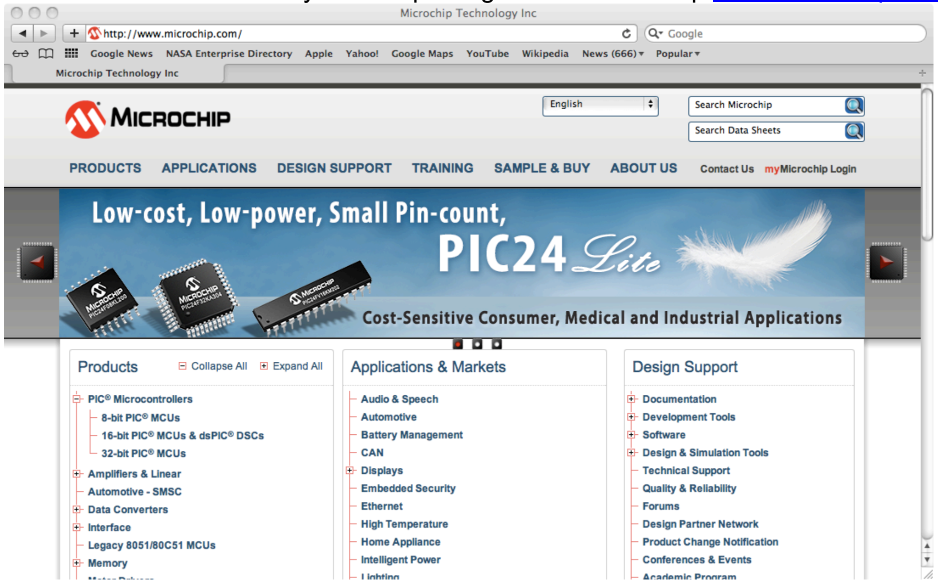
Part 5A: Extra Credit (10 points)
Update or write your own version of a PIC code generator and edit the hw12_test.sh file and remove the comments to test the other mathXX.txt files. Attempt to support as many test files as possible. When the extra credit code is complete and working then execute the homework script file "./hw12_test.sh" using the command below.
./hw12_test.sh &> hw12_test.txt

Print out this version of the **hw12_test.txt** file. Mark this output with **EXTRA CREDIT** and put your name, assignment number, date on it and turn it in instead of the file from Part 4. Upload **ONLY** your updated/new code generator file to blackboard in the assignment area. Submit it with the comments **hw12** with your **CaseID** on the title line. The extra credit will be graded on-line and your score will be added to your assignment score.

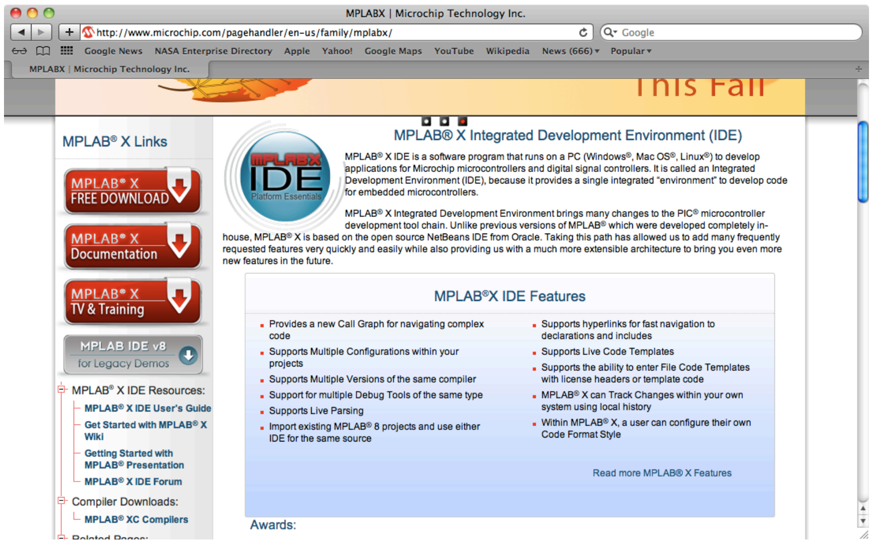
Part 5B: Extra Credit (5 points)
Test the output of your compiler using MPLAB, public domain software for the *Microchip PIC* microprocessor. Install MPLAB on your computer or use one of the computers in the lab. If you have not used MPLAB, you will need a copy of the *MPLAB* User Guide, Part 2 – MPLAB IDE Tutorial. You can download a version of the guide from https://blackboard.case.edu or from the Microchip website.

The software is designed as a standard Integrated Development Software (IDE) application. As an engineer you will use a number of different IDEs in your career. Most IDE applications have the same type of operations. You typically download the software to a target computer and install the applications. You may need to register the software with the company. Most companies allow their software to be downloaded and used for free but with reduced functionality. When you need to upgrade to the full version you purchase the software. Otherwise you are free to use the software in your education, research or development.

Step 1. [Optional step] Install the software
To install the software on your computer go to the Microchip www.microchip.com website.



In the menu bar under **DESIGN SUPPORT** click on the **MPLAB X IDE** link. Below the three MPLAB X buttons find the **MPLAB IDE v8 for Legacy Demos** button and click on the link.


MPLABX | Microchip Technology Inc.
http://www.microchip.com/pagehandler/en-us/family/mplabx/
This Fall
MPLAB® X Links
MPLAB® X FREE DOWNLOAD
MPLAB® X Documentation
MPLAB® X TV & Training
MPLAB IDE v8 for Legacy Demos
MPLAB® X IDE Resources:
MPLAB® X IDE User's Guide
Get Started with MPLAB® X Wiki
Getting Started with MPLAB® Presentation
MPLAB® X IDE Forum
Compiler Downloads:
MPLAB® XC Compilers
MPLAB X IDE Platform Essentials

MPLAB® X Integrated Development Environment (IDE)

MPLAB® X IDE is a software program that runs on a PC (Windows®, Mac OS®, Linux®) to develop applications for Microchip microcontrollers and digital signal controllers. It is called an Integrated Development Environment (IDE), because it provides a single integrated "environment" to develop code for embedded microcontrollers.

MPLAB® X Integrated Development Environment brings many changes to the PIC® microcontroller development tool chain. Unlike previous versions of MPLAB® which were developed completely in-house, MPLAB® X is based on the open source NetBeans IDE from Oracle. Taking this path has allowed us to add many frequently requested features very quickly and easily while also providing us with a much more extensible architecture to bring you even more new features in the future.

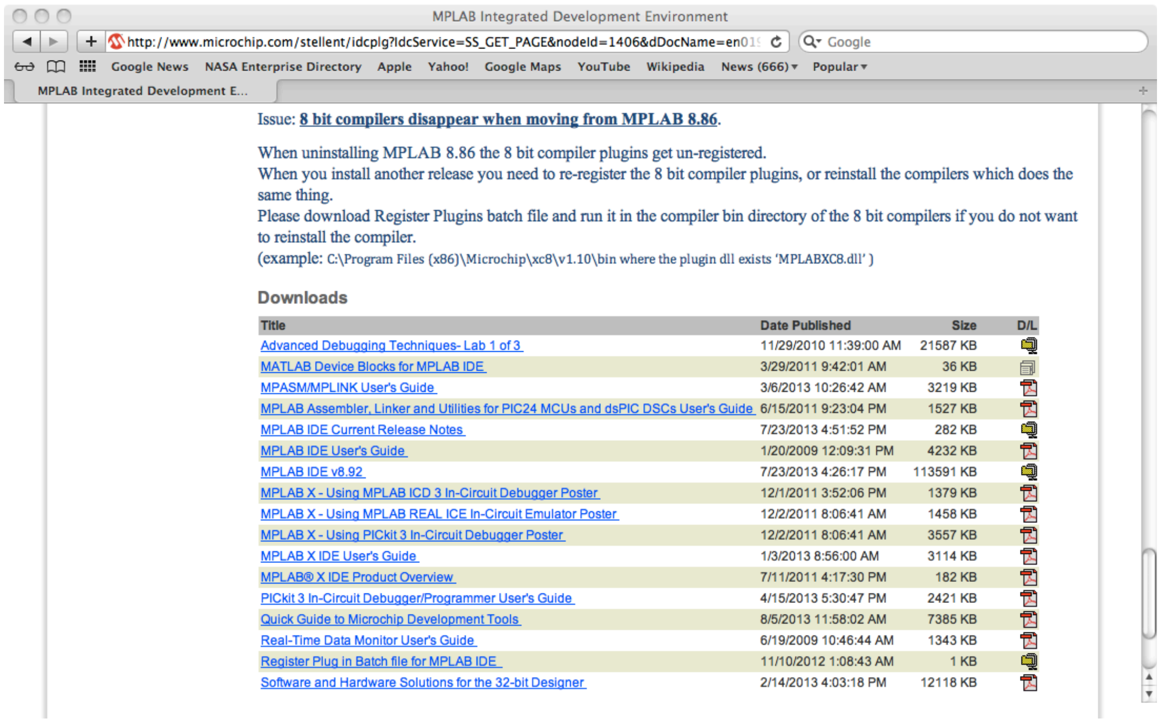
MPLAB®X IDE Features

- Provides a new Call Graph for navigating complex code
- Supports Multiple Configurations within your projects
- Supports Multiple Versions of the same compiler
- Support for multiple Debug Tools of the same type
- Supports Live Parsing
- Import existing MPLAB® 8 projects and use either IDE for the same source
- Supports hyperlinks for fast navigation to declarations and includes
- Supports Live Code Templates
- Supports the ability to enter File Code Templates with license headers or template code
- MPLAB® X can Track Changes within your own system using local history
- Within MPLAB® X, a user can configure their own Code Format Style

Read more MPLAB® X Features

Awards:

On the next page scroll to the bottom and click on the MPLAB IDE v8.92 link and download the zip file. Then click on the MPLAB IDE User's Guide and download the file.



MPLAB Integrated Development Environment

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en01

MPLAB Integrated Development E...

Issue: **8 bit compilers disappear when moving from MPLAB 8.86**.

When uninstalling MPLAB 8.86 the 8 bit compiler plugins get un-registered.
When you install another release you need to re-register the 8 bit compiler plugins, or reinstall the compilers which does the same thing.
Please download Register Plugins batch file and run it in the compiler bin directory of the 8 bit compilers if you do not want to reinstall the compiler.
(example: C:\Program Files (x86)\Microchip\xc8\v1.10\bin where the plugin dll exists 'MPLABXC8.dll')

Downloads

Title	Date Published	Size	D/L
Advanced Debugging Techniques- Lab 1 of 3	11/29/2010 11:39:00 AM	21587 KB	
MATLAB Device Blocks for MPLAB IDE	3/29/2011 9:42:01 AM	36 KB	
MPASM/MPLINK User's Guide	3/6/2013 10:26:42 AM	3219 KB	
MPLAB Assembler, Linker and Utilities for PIC24 MCUs and dsPIC DSCs User's Guide	6/15/2011 9:23:04 PM	1527 KB	
MPLAB IDE Current Release Notes	7/23/2013 4:51:52 PM	282 KB	
MPLAB IDE User's Guide	1/20/2009 12:09:31 PM	4232 KB	
MPLAB IDE v8.92	7/23/2013 4:26:17 PM	113591 KB	
MPLAB X - Using MPLAB ICD 3 In-Circuit Debugger Poster	12/1/2011 3:52:06 PM	1379 KB	
MPLAB X - Using MPLAB REAL ICE In-Circuit Emulator Poster	12/2/2011 8:06:41 AM	1458 KB	
MPLAB X - Using PICkit 3 In-Circuit Debugger Poster	12/2/2011 8:06:41 AM	3557 KB	
MPLAB X IDE User's Guide	1/3/2013 8:56:00 AM	3114 KB	
MPLAB® X IDE Product Overview	7/11/2011 4:17:30 PM	182 KB	
PICkit 3 In-Circuit Debugger/Programmer User's Guide	4/15/2013 5:30:47 PM	2421 KB	
Quick Guide to Microchip Development Tools	8/5/2013 11:58:02 AM	7385 KB	
Real-Time Data Monitor User's Guide	6/19/2009 10:46:44 AM	1343 KB	
Register Plug in Batch file for MPLAB IDE	11/10/2012 1:08:43 AM	1 KB	
Software and Hardware Solutions for the 32-bit Designer	2/14/2013 4:03:18 PM	12118 KB	

Unzip the MPLAB_IDE_8_92.zip file. Change directory into the MPLAB_IDE_8_92 and double click the setup.exe. Open the User's Guide and follow Part 2 – 4.2 MPLAB IDE FEATURES AND INSTALLATION. Follow the instructions to install the software onto your computer. You may need to reboot the computer after installing the software.

Step 2. Work through the tutorial

Execute *MPLAB*. Follow the steps in Part 4.2.2 of the MPLAB IDE Tutorial and when using the **Project Wizard** use the following information:

Step One: Select a device: and pick PIC16F1827. Click Next.
Step Two: Select a language toolsuite: Use the Microchip MPASM Assembler (mpasmwin.exe) v5.51. It should be listed under: C:\Program Files\Microchip\MPASM Suite\MPASMWIN.exe. Click Next.
Step Three: Create New Project File: Use the Browse… button to change to your EECS337/caseid/hw11/ directory using your Case ID. Then enter the file name (Design12). Click Next.
Step Four: Add the Math24.asm the file to your project. Click Finish to create the work space.
Step Five: Now build the assembler program "Project>Build All" function. Pick Absolute code, if it ask you. You should see the output window have no build errors and then you can continue on with the lab.

Under the Debugger->Select Tool pick the MPLAB SIM (simulator) version. The debug buttons should now be part of Run the program using the "Animate" button (two arrows) and watch the instructions execute. Press the "Reset" button (page loop) and watch the process jump to the start of program memory. You can stop execution (pause), set break points and run (single arrow) the program. View the register files and print out a window snap shot showing your code running inside the software.

Print out this snap shot file showing your program execution. Mark this output with **EXTRA CREDIT** and put your name, assignment number, date on it and turn it in. Answer the questions at the end of the assignment.

Part 6: Laboratory Questions (do any 4 questions)

1) If you wrote your own code generator then explain the code you designed.

2) If you changed the hw12_test.sh file then explain how many extra files you were able to support.

3) Change the symbol table to use the CBLOCK ENDC instructions instead of the EQU statements. See the example below.

```
cblock 0x20 ; define gpr variable register locations
        myvar1  ; user variables allocated contiguously
        myvar2  ;
        myvar3  ;
    endc

sample1        equu    0x7d        ; sample user registers
sample2        equ     0x7e        ;
sample3        equ     0x7f        ;
```

4) Update the quad to tuple function to support the array type. Explain the code you designed and the change in the output.

5) Identify and fix a bug in the current software. Explain the code you designed and the how it changed the output.

6) Identify and fix another bug in the current software. Explain the code you designed and the how it changed the output.

7) Add a new instruction into the current code generator. Explain the code you designed and the change in the output.

8) Add another new instruction into the current code generator. Explain the code you designed and the change in the output.