Tutorial

Essentially all the Blackfin instruction you need for all of ENCM511.

The instructions are easy. It's knowing when to use them that is the difficult part of the course.

Big hints for Lab. 1

- Write a Blackfin assembly language routine that does not return until Switch 1 is up (switch set, 1), switch 2 is down (switch clear, 0) and switch 4 is set.

- Part of the code that you will use in Laboratory 1 to store commands for the operation of the radio controlled car.

Burglar alarm – second example

Very suitable for quizzes and midterm

Light sensor output is 1 if there is light shining on the sensor.

Write a Blackfin Assembly Subroutine that waits until somebody leaves or enters the room, and then returns the value 1 or 2 depending on how many people leave the room at the same time.

Essentially all the instructions you ever need to know on ANY processor.

- assignment =
- add (+), subtract (-)
- bitwise AND (&), bit wise OR (|)
- Shift values left and right (arithmetic and logical)
- store a value to memory
- retrieve, read, load a value from memory
- Handle 8-bit, 16-bit and 32-bit DATA values
- do an if statement and then (conditionally jump)
- Call a function, return from a function
Basic assembly code operations on Blackfin processor -- ADD

- Place a 1 into register R0
- Place a 2 into register R1
- Add the values and store in R3 and return the value to a C / C++ program

```
.section program;
.global AddASM;
1. AddASM:      // Assembler: This label and next instruction as ON SAME LINE
    R0 = 1;
2. R1 = 2;
3. R3 = R1 + R2;   // DO CODE REVIEW -- Note typo -- WHERE
                     // Participate in HUMAN CLICKER
                      Left hand Line 1; Left and Right Line 2; Left and Right and Mouth surprised (0)
                      Line 3; Left and Right and Mouth surprised (0) and stand-up Line 4 or 5
4. R0 = R3;       // C++ REQUIREMENT TO RETURN VALUE IN R0
5. AddASM.END:   // Assembler: This label and next instruction as ON SAME LINE
                  // Expected Result (when testing) is 3
RTS;
```

Basic assembly code operations on Blackfin processor -- SUB

- Place a 1 into register R0
- Place a 2 into register R1
- Subtract R0 from R1 and store in R3 and return the value to a C / C++ program

```
.section program;
.global SubASM;
SubASM:
    R0 = 1;
    R1 = 2;
    R3 = R1 – R0;
4. R0 = R3;       // C++ REQUIREMENT, RETURN VALUE IN R0
5. SubASM.END:   // Expected Result is 1
RTS;
```

UNPLANNED HIDDEN ERROR
YOU CAN’T USE REGISTER R4

- Place a 1 into register R0
- Place a 2 into register R1
- Subtract R0 from R1 and store in R4 and return the value to a C / C++ program

```
.section program;
.global SubASM;
SubASM:
    R0 = 1;
    R1 = 2;
    R4 = R1 – R0;               // OTHER PARTS OF YOUR C CODE IS USING R4
4. R0 = R4;       // C++ REQUIREMENT TO RETURN VALUE IN R0
SubASM.END:       // Expect Result is 1
RTS;
```

CORRECTED CODE FOR USING R4 (NO-VOLATILE) REGISTER

- Place a 1 into register R0
- Place a 2 into register R1
- Subtract R0 from R1 and store in R4 and return the value to a C / C++ program

```
.section program;
.global SubASM;
SubASM:
    [--SP] = R4;   // SAVE THE VALUE THE OTHER C CODE USES
                      // JITK – IMPORTANT: --SP MEANS PRE-DECREMENT SP THEN USE IT
                      // SP -- MEANS USE SP THEN POST DECMENT  (C and ASM)
    R0 = 1;
    R1 = 2;
    R4 = R1 – R0;               // OTHER PARTS OF YOUR C CODE IS USING R4
4. R0 = R4;       // C++ REQUIREMENT TO RETURN VALUE IN R0
                // JITK – IMPORTANT: SP++ MEANS USE SP AND POST-INCREMENT
                // ++SP MEANS PRE-INCREMENT SP AND THEN USE IT  (C and ASM)
SubASM.END:       // Expect Result is 1
RTS;
```
JITK -- Learn this syntax NOW and avoid many wasted hours in the labs (and lost (easy) marks in quizzes / exams)

section program;
.global SubASM;

AN APPLE AND PEAR AND APPLES AND ORANGES INSTRUCTION

Subasm:

[--SP] = R4;

// SAVE THE VALUE THE OTHER C CODE USES
// IMPORTANT --SP NOT SP--
// --SP MEANS PRE-DECREMENT SP THEN USE IT
// AS AN ADDRESS POINTER TO SAVE A
// VALUE TO MEMORY -- DIFFERENT THAN SP--

R4 = [SP++];

// SAVE THE VALUE THE OTHER C CODE USES

Subasm.END:

RTS;

Embedded UnitTest TEST FORMAT

Design requirement -- AddASM( ) does the following
- Place a 1 into register R0, Place a 2 into register R1
- Adds the values and store in R3 and return the value (3) to a C / C++ program in R0

Design requirement -- SubASM( ) does the following
- Place a 1 into register R0, Place a 2 into register R1
- Subtracts R0 from R1, Place a 2 into register R1
- Subtracts R0 from R1 and store in R3 and return the value to a C / C++ program in R0

extern "C" int AddASM( void);
TEST(AddASM) {
    int expectedValue = 3;
    CHECK_VALUES(expectedValue, AddASM( ) );
}

extern "C" int SubASM( void);
TEST(SubASM) {
    int expectedValue = 1;
    CHECK_VALUES(expectedValue, SubASM( ) );
}

Basic assembly code operations on Blackfin processor -- AND

- Place a 1 into register R0
- Place a 3 into register R1
- AND the values and store in R3 and return the value to a C / C++ program

AndAsm:

R0 = 1;                            R0 (last 8 bits) = b0000 0001
R1 = 3; R1 (last 8 bits) = b0000 0011 R3 = R1 & R0;                R3 (last 8 bits) = b???? ???? – Know for E-UNIT test
R0 = R3; // C++ REQUIREMENT FOR RETURN VALUE

AndAsm.END:

RTS; // Expect Result is 1 & 3 = ????

Introduce the Blackfin GPIO register
Input / Output Interface to the outside world
Now using tutorial to implement Lab. 1

Memory mapped 16-bit register

Name FIO_FLAG_D defined in <blackfin.h>

Figure 14-2. Flag Data Register
**TASK:** -- Read the 16 bit value in the Blackfin GPIO data register and return value to “C++”

**E.G. Switch values, thermal sensor values**

Non self-documented code (poor for getting marks)

Blackfin manual information

```
#include <blackfin.h>
.section program;
.global ReadGPIO;
ReadGPIO:
#define FLAG_MEMORY_LOCATION 0xFFC00700
P0.L = lo(FLAG_MEMORY_LOCATION); P0.H = hi(FLAG_MEMORY_LOCATION);
R3 = W[P0] (Z); // Read 16-bit value,
// Extend with 0's to 32-bits and store in 32 bit register
// Easier code to remember how to use (and less likely to make a typo)
P0.L = lo(FIO_FLAG_D); P0.H = hi(FIO_FLAG_D); // DEFINED IN <blackfin.h>
R3 = W[P0] (Z); // Read 16-bit bit pattern value and zero extend it to 32 bits
// JITK – Extension using (Z) means that C++ code will not get confused by (unknown) top bits
// that were filled with garbage from the last function (C++ or ASM) sharing your use of R3

R0 = R3; // C++ REQUIREMENT TO RETURN VALUE IN R0
ReadGPIO.END:
RTS; // Expect Result is ??????
```

**Real life problem with a real processor**

What happens if you (unintentionally) use this code?

```
#include <blackfin.h>
.section program;
.global ReadGPIO;
ReadGPIO:
// Easier code to remember how to use (and less likely to make a typo)
P0.L = lo(FIO_FLAG_D); P0.H = hi(FIO_FLAG_D);
R3 = W[P0] (Z); // Read 16-bit bit pattern value and zero extend it to 32 bits
R3 = [P0] ; // Un-intentional read of 32 bits rather than 16
R0 = R3; // C++ REQUIREMENT TO RETURN VALUE IN R0
ReadGPIO.END:
RTS; // Expect Result is ??????
```

**PROCESSOR COULD DO ONE OF THE FOLLOWING WHEN MIS-READING A PERIPHERAL REGISTER**

A) CRASH – because the FIO_FLAG_D address logic will not handle 32 bit reads
   THIS IS THE BEST CASE – YOU KNOW SOMETHING IS WRONG
   GET – UNEXPECTED EXCEPTION ERROR MESSAGE FROM VDSP IDDE
   IN THE FIELD – BLOWS UP THE PLANT

B) GIVES BACK (READS) THE WRONG VALUE

C) SOMETIMES GIVES BACK THE WRONG VALUE
   EVEN A WORSE CASE, YOUR TEST MIGHT SOMETIMES WORK (FAILS THIRD TIME)

**TASK:** -- Return ONLY bit 11 of Blackfin GPIO data and return value to “C++”. This is connected to SWITCH 3 (Hidden code defect)

Non self-documented code (poor for getting marks)

```
#include <blackfin.h>
.section program;
.global ReadGPIOBit;
ReadGPIOBit:
#define FLAG_MEMORY_LOCATION 0xFFC00700
P0.L = lo(FLAG_MEMORY_LOCATION); P0.H = hi(FLAG_MEMORY_LOCATION);
R3 = W[P0] (Z); // Read 16-bit value,
// Extend with 0's to 32-bits and store in 32 bit register
// Easier code to remember how to use (and less likely to make a typo)
P0.L = lo(FIO_FLAG_D); P0.H = hi(FIO_FLAG_D); // DEFINED IN <blackfin.h>
R3 = W[P0] (Z); // Read 16-bit bit pattern value and zero extend it to 32 bits
// JITK – Extension using (Z) means that C++ code will not get confused by (unknown) top bits
// that were filled with garbage from the last function (C++ or ASM) sharing your use of R3

R2 = BIT11MASK; // Binary pattern 0000 1000 0000 0000
R0 = R3 & R2; // Return bit 11 in register 0 – IGNORE OTHER VALUES IN OTHER BITS
ReadGPIOBit.END:
RTS; // Expected Result is 0x0800 if button SW3 pressed
```

**TASK:** -- Return bit 9 (SW2) ONLY if bit 11 of Blackfin GPIO data register (SW4) is also pressed

**DEMONSTRATE Do-WHILE LOOP FOR ACCEPTING INPUT IN LAB 1**

```
#include <blackfin.h>
.section program;
.global ReturnOnBit11;
ReturnOnBit11:
#define BIT11MASK 0x0800    // Binary pattern 0000 1000 0000 0000 (BIT 11 IS SET IN MASK)
R2 = BIT11MASK; // Changed register name to R1
DO_WHILE:
CC = R1 == R2; // CC condition code SET to 1 if R1 == R2 (bit 11 set) and 0 otherwise
IF CC JUMP ReturnOnBit11; // Jumps to start of routine and runs the code again
#define BIT9MASK 0x0200    // Binary pattern 0000 0010 0000 0000
R2 = BIT9MASK;
R0 = R3 & R2; // Return bit 9 in register 0 – IGNORE OTHER VALUES IN OTHER BITS
ReturnOnBit11.END:
RTS;
```
Why is this code “wrong”? Is the “wrongness” important?

```c
#include <blackfin.h>

.section program; .global ReturnOnBit11;
ReturnOnBit11: // Easier code to remember how to use (and less likely to make a typo)
P0.L = lo(FIO_FLAG_D); P0.H = hi(FIO_FLAG_D);
R3 = W[P0] (Z); // Read 16-bit value,
// Extend with 0’s to 32-bits and store in 32 bit register

// Turn all bits but bit 11 into zero
#define BIT11MASK 0x0800 // Binary pattern 0000 1000 0000 0000 (BIT 11 IS SET)
R2 = BIT11MASK;
R1 = R3 & R2; // Changed register name to R1
CC = R1 == R2; // CC condition code SET to 1 if R1 == R2 (bit 11 set) and 0 otherwise
IF CC JUMP ReturnOnBit11;

#define BIT9MASK 0x0200 // Binary pattern 0000 1000 0000 0000 (BIT 11 IS SET)
R2 = BIT9MASK;
R3 = W[P0] (Z); // Read the input data register again
R0 = R3 & R2; // Return bit 9 in register 0
ReturnOnBit11.END:
RTS;
```

NEW INSTRUCTION – NOT LIKE MIPS

Return a 1 if bit 11 of Blackfin GPIO data register (SW4) is set and return 0 otherwise

Non self-documented code (poor for getting marks)

```c
#include <blackfin.h>

.section program; .global Return1OnBit11;
Return1OnBit11: // Easier code to remember how to use (and less likely to make a typo)
P0.L = lo(FIO_FLAG_D); P0.H = hi(FIO_FLAG_D);
R3 = W[P0] (Z); // Read 16-bit value,
// Extend with 0’s to 32-bits and store in 32 bit register

// Turn all bits but bit 11 into zero
#define BIT11MASK 0x08000 // Binary pattern 0000 1000 0000 0000
R2 = BIT11MASK;
R1 = R3 & R2; // C++ REQUIREMENT
CC = R1 == R2; // CC condition code 1 if R1 == R2 (bit 11 set) and 0 otherwise
IF !CC R0 = 0; // If condition not true, return a 0 (in R0)
IF CC R0 = 1; // If condition true, return a 1 (in R0)

// Is this syntax really possible? Write simple code to check
Return1OnBit11.END:
RTS;
```

Add a trial.asm file to Familiarization Lab. code to see if the syntax is correct – ITS NOT

This version of an if instruction syntax is possible but pointless to use in this situation -- if CC reg = reg
This code assembles – but is it correct? (meaning does it give the expected answer)?

- Read Chapter 10 of programming manual

So complicated – why bother to code this way? Difficult to maintain this type of coding – KIS instead

Tutorial Exercise

Switches 0, 1, 2 and 3 are connected to pins 8, 9, 10, and 11 respectively

Pins 0 to 7 are connected to the Blackfin video channel (input)
Pins 12 to 15 are connected to ????

- Write a Blackfin assembly language routine that does not return until Switch 1 is up (set, 1), switch 2 is down (clear, 0) and switch 4 is set.

- Part of the code that you will use in Laboratory 1 to store commands for the operation of the radio controlled car.

Burglar alarm – second example

Very suitable for quizzes and midterm

Light sensor output is 1 if there is light shining on the sensor.
Write a Blackfin Assembly Subroutine that waits until somebody leaves or enters the room, and then returns the value 1 or 2 depending on how many people leave the room at the same time
Subroutine to call Return1onBit11( ) and store result in memory

```c
#include <blackfin.h>

.section L1_data;
.byte2 _result;

.section program;
.extern Return1OnBit11; // Code in another file
.global NewFunction;
NewFunction:
  LINK 16;
  CALL Return1OnBit11; // Returns value in R0 ALWAYS
  P0.l = lo(_result); P0.h = hi(_result);
  // value returned in R0 by convention
  W[P0] = R0;
  UNLINK;
NewFunction.END:
  RTS;
```

This code looks the same – but is the difference between a B grade and a D grade and many hours wasted in the lab.

```c
#include <blackfin.h>

.section L1_data;
.byte2 _result;

.section program;
.extern Return1OnBit11; // Code in another file
.global NewFunction;
NewFunction:
  LINK 16;
  P0.l = lo(_result); P0.h = hi(_result);
  CALL Return1OnBit11; // Returns value in R0 ALWAYS
  // P0.l = lo(_result); P0.h = hi(_result);
  // value returned in R0 by convention
  W[P0] = R0;
  UNLINK;
NewFunction.END:
  RTS;
```

WHAT DOES THIS CODE FAIL?

What’s the hidden error
- I either put this error deliberately in the notes (so see if people awake) or accidently used it
- I said -- **Switches SW0, SW1, SW2 and SW3 are connected to pins 8, 9, 10, and 11 respectively**
- CHECK THE FRONT PANEL
  Switches connected to pins 8, 9, 10, and 11 labeled as SW1, SW2, SW3 and SW4
- Remember array elements are numbered starting at 0 in C and from 1 in Matlab!! – So its not just me!!

Essentially all the instructions you ever need to know on a processor.
- assignment =
- add (+), subtract (-)
- bitwise and (&), bit wise or (|)
- Shift values left and right (arithmetic and logical)
- store a value to memory
- retrieve, read, load a value from memory
- Handle 8-bit, 16-bit and 32-bit DATA values
- do an if statement and then (conditionally jump)
- Call a function, return from a function