CEG2400 Keil examples:

PLL setup of startup code (Lec 5)*

- F_{OSC}= 11.0592MHz
- PLLCFG_Val = 0x24
  - MSEL=4 (M=MSEL+1=5)
  - CCLK= M * Fosc=55.296MHz
- P
  - PSEL=1 i.e. P=2
  - (low) P= 156 MHz / (2 x 55.296 MHz) =1.41
  - (high) P= 320 MHz / (2 x 55.296 MHz) =2.89
- PCLK=CCLK/4=13.824MHz

Lec 8: irqhello debugging

- Keil allows checking of all registers, interrupt breakpoints, etc
- Very powerful tool to check understanding of LPC21xx and for debugging

CEG2400 Lecture 9: Passing Parameters and C

Source:
http://www.intel.com/education/highered/Embedded/Lectures.htm

This Lecture

- Assembly Code from C Programs (7 Examples)
  - Compiler often optimizes out redundant operations
  - Need to be able to understand compiler generated code
  - Also aids in writing good assembly language
  - These examples take some effort to understand!
- Dealing With Structures
- Interfacing C Code with Intel Xscale® Assembly
- Intel Xscale® libraries and `armsd`

The Structure of an Assembler Module

Chunks of code or data manipulated by the linker

Minimum required block (why?)

```
        AREA Example, CODE, READONLY  ; name of code block
        ENTRY                         ; 1st exec. instruction
start
        MOV     r0, #15               ; set up parameters
        MOV     r1, #20
        BL      func                  ; call subroutine
        SWI     0x11                  ; terminate program
func                                  ; the subroutine
        ADD     r0, r0, r1            ; r0 = r0 + r1
        MOV     pc, lr                ; return from subroutine
                                      ; result in r0
        END                           ; end of code
```

First instruction to be executed

Example 1: A Simple Program

```
int a,b;
int main()
{
   a = 3;
   b = 4;
} /* end main() */
```

```
        AREA ||.text||, CODE, READONLY
main PROC
|L1.0|
        LDR      r0,|L1.28|
        MOV      r1,#3
        STR      r1,[r0,#0]  ; a
        MOV      r1,#4
        STR      r1,[r0,#4]  ; b
        MOV      r0,#0
        BX       lr   // subroutine call
|L1.28|
        DCD      ||.bss$2||
        ENDP
        AREA ||.bss||
a
||.bss$2||
        DCD    0
b
        DCD    0
        EXPORT main
        EXPORT b
        EXPORT a
        END
```

label "L1.28" - compiler tends to make the labels equal to the address

declare one or more words

loader will put the address of |||.bss$2| into this memory location

declares storage (1 32-bit word) and initializes it with zero

Example 1 (cont'd)

```
address

0x00000000
0x00000004
0x00000008
0x0000000C
0x00000010
0x00000014
0x00000018
0x0000001C

0x00000020

0x00000024
```

```
        AREA ||.text||, CODE, READONLY
main PROC
|L1.0|
        LDR      r0,|L1.28|
        MOV      r1,#3
        STR      r1,[r0,#0]  ; a
        MOV      r1,#4
        STR      r1,[r0,#4]  ; b
        MOV      r0,#0
        BX       lr   // subroutine call
|L1.28|
        DCD      0x00000020
        ENDP
        AREA ||.bss||
a
||.bss$2||

        DCD 00000000
b

        DCD 00000000
        EXPORT main
        EXPORT b
        EXPORT a
        END
```

This is a pointer to the |x$dataseg| location

Notes on Example 1

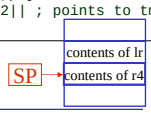
- a and b are global variables
- |L1.28| contains &a
- Compiler knows that &b = &a+4

Example 2: Calling A Function

```
int tmp;
void swap(int a, int b);
int main()
{
   int a,b;
   a = 3;
   b = 4;
   swap(a,b);
} /* end main() */

void swap(int a,int b)
{
   tmp = a;
   a = b;
   b = tmp;
} /* end swap() */
```

```
          AREA ||.text||, CODE, READONLY
swap PROC
        LDR      r2,|L1.56|
        STR      r0,[r2,#0]  ; tmp
        MOV      r0,r1
        LDR      r2,|L1.56|
        LDR      r1,[r2,#0]  ; tmp
        BX       lr
main PROC
        STMFD    sp!,{r4,lr}
        MOV      r3,#3
        MOV      r4,#4
        MOV      r1,r4
        MOV      r0,r3
        BL       swap
        MOV      r0,#0
        LDMFD    sp!,{r4,pc}
|L1.56| DCD      ||.bss$2|| ; points to tmp
        END
```

STMFD - store multiple, full descending
sp ← sp - 4
mem[sp] = lr ; linkreg
sp ← sp – 4
mem[sp] = r4 ; linkreg



Notes on Example 2

- a and b are local variables and are in r3 and r4
- |L1.56| points to tmp
- a and b passed in r0 and r1 to swap()

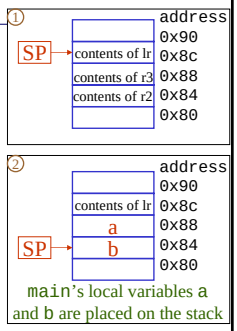
Example 3: Manipulating Pointers

```
int tmp;
int *pa, *pb;
void swap(int a, int b);
int main()
{
   int a,b;
   pa = &a;
   pb = &b;
   *pa = 3;
   *pb = 4;
   swap(*pa, *pb);
} /* end main() */

void swap(int a,int b)
{
   tmp = a;
   a = b;
   b = tmp;
} /* end swap() */
```

```
        AREA ||.text||, CODE, READONLY
swap    LDR      r1,|L1.60| ; get tmp addr
        STR      r0,[r1,#0] ; tmp = a
        BX       lr
main    STMFD    sp!,{r2,r3,lr}
        LDR      r0,|L1.60| ; get tmp addr
        ADD      r1,sp,#4   ; &a on stack
        STR      r1,[r0,#4] ; pa = &a
        STR      sp,[r0,#8] ; pb = &b (sp)
        MOV      r0,#3
        STR      r0,[sp,#4] ; *pa = 3
        MOV      r1,#4
        STR      r1,[sp,#0] ; *pb = 4
        BL       swap       ; call swap
        MOV      r0,#0
        LDMFD    sp!,{r2,r3,pc}
|L1.60| DCD      ||.bss$2||
        AREA ||.bss||
||.bss$2||
        tmp   DCD 00000000
        pa    DCD 00000000
        pb    DCD 00000000
```

Example 3 (cont'd)

```
AREA ||.text||, CODE, READONLY
swap    LDR     r1,|L1.60|
        STR     r0,[r1,#0]
        BX      lr
main    STMFD   sp!,{r2,r3,lr}  ①
        LDR     r0,|L1.60|  ; get tmp addr
        ADD     r1,sp,#4    ; &a on stack
        STR     r1,[r0,#4]  ; pa = &a      ②
        STR     sp,[r0,#8]  ; pb = &b (sp)
        MOV     r0,#3
        STR     r0,[sp,#4]
        MOV     r1,#4
        STR     r1,[sp,#0]
        BL      swap
        MOV     r0,#0
        LDMFD   sp!,{r2,r3,pc}
|L1.60| DCD     ||.bss$2||
        AREA ||.bss
||.bss$2||
        tmp   DCD 00000000
        pa    DCD 00000000   ; tmp addr + 4

        pb    DCD 00000000   ; tmp addr + 8
```



Notes on Example 3

- |L1.60| contains address of tmp
- a and b are local variables stored at locations sp+4 and sp respectively in main()
- pa and pb are global variables computed as offsets from &tmp (pa=&tmp+4, pb=&tmp+8)
- pa is set to &a
- pb is set to &b
- Since a and b are changed but not used again in swap(), the code is removed by the compiler so swap() only contains tmp=a

Example 4: Dealing with "`struct`"s

```
typedef struct
testStruct {
   unsigned int a;
   unsigned int b;
   char c;
} testStruct;

testStruct *ptest;

int main()
{
   ptest->a = 4;
   ptest->b = 10;
   ptest->c = 'A';
} /* end main() */
```

```
AREA ||.text||, CODE, READONLY
main PROC      r1 ← M[#L1.56] is the pointer to ptest
|L1.0|
        MOV     r0,#4       ; r0 ← 4
        LDR     r1,|L1.56|
        LDR     r1,[r1,#0]  ; r1 ← &ptest
        STR     r0,[r1,#0]  ; ptest->a = 4
        MOV     r0,#0xa     ; r0 ← 10
        LDR     r1,|L1.56|
        LDR     r1,[r1,#0]  ; r1 ← ptest
        STR     r0,[r1,#4]  ; ptest->b = 10
        MOV     r0,#0x41    ; r0 ← 'A'
        LDR     r1,|L1.56|
        LDR     r1,[r1,#0]  ; r1 ← &ptest
        STRB    r0,[r1,#8]  ; ptest->c = 'A'
        MOV     r0,#0
        BX      lr          ← watch out, ptest is only a ptr
                              the structure was never malloc'd!
|L1.56|
        DCD     ||.bss$2||
        AREA ||.bss||
ptest
||.bss$2||
        % 4
```


Notes on Example 4

- |L1.56| is a pointer to the structure this pointer is stored in r1
- Elements of the structure are accessed as offsets from r1
  - e.g. STR r0,[r1,#4] **; ptest->b = 10**

Example 5: Dealing with Lots of Arguments

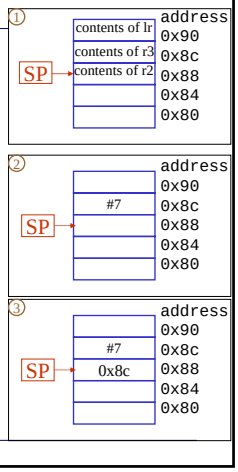
```
int tmp;
void test(int a, int b, int
    c, int d, int *e);
int main()
{ int a, b, c, d, e;
  a = 3;
  b = 4;
  c = 5;
  d = 6;
  e = 7;
  test(a, b, c, d, &e);
} /* end main() */

void test(int a,int b,
  int c, int d, int *e)
{
  tmp = a;
  a = b;
  b = tmp;
  c = b;
  b = d;
  *e = d;
} /* end test() */
```

```
        AREA ||.text||, CODE, READONLY
test    LDR     r1,[sp,#0]  ; get &e
        LDR     r2,|L1.72|  ; get tmp addr
        STR     r0,[r2,#0]  ; tmp = a
        STR     r3,[r1,#0]  ; *e = d
        BX      lr
main PROC
        STMFD   sp!,{r2,r3,lr} ; → 2 slots
        MOV     r0,#3   ; 1st param a
        MOV     r1,#4   ; 2nd param b
        MOV     r2,#5   ; 3rd param c
        MOV     r12,#6  ; 4th param d
        MOV     r3,#7   ; overflow → stack
        STR     r3,[sp,#4] ; e on stack
        ADD     r3,sp,#4
        STR     r3,[sp,#0] ; &e on stack
        MOV     r3,r12  ; 4th param d in r3
        BL      test
        MOV     r0,#0   ← r0 holds the return value
        LDMFD   sp!,{r2,r3,pc}
|L1.72|
        DCD     ||.bss$2||
tmp
```


Example 5 (cont'd)

```
AREA ||.text||, CODE, READONLY
test    LDR     r1,[sp,#0]  ; get &e
        LDR     r2,|L1.72|  ; get tmp addr
        STR     r0,[r2,#0]  ; tmp = a
        STR     r3,[r1,#0]  ; *e = d
        BX      lr
main PROC
        STMFD   sp!,{r2,r3,lr} ; → 2 slots ①
        MOV     r0,#3   ; 1st param a
        MOV     r1,#4   ; 2nd param b
        MOV     r2,#5   ; 3rd param c
        MOV     r12,#6  ; 4th param d
        MOV     r3,#7   ; overflow → stack
        STR     r3,[sp,#4] ; e on stack    ②
        ADD     r3,sp,#4
        STR     r3,[sp,#0] ; &e on stack   ③
        MOV     r3,r12  ; 4th param d in r3
        BL      test
        MOV     r0,#0
        LDMFD   sp!,{r2,r3,pc}
|L1.72|
        DCD     ||.bss$2||
tmp
```

Note: In "test", the compiler removed the assignments to a, b, and c -- these assignments have no effect, so they were removed



Notes on Example 5

- In main() e is located at sp+4, &e at sp
- a, b, c, d are passed to test() is r0, r1, r2, r3
- test() only contains code that effects tmp and *e
  - Other code has no effect so is omitted

Example 6: Nested Function Calls

```
int tmp;
int swap(int a, int b);
void swap2(int a, int b);
int main(){
   int a, b, c;
   a = 3;
   b = 4;
   c = swap(a,b);
} /* end main() */

int swap(int a,int b){
   tmp = a;
   a = b;
   b = tmp;
   swap2(a,b);
   return(10);
} /* end swap() */

void swap2(int a,int b){
   tmp = a;
   a = b;
   b = tmp;
} /* end swap() */
```

```
swap2   LDR     r1,|L1.72|
        STR     r0,[r1,#0]  ; tmp ← a
        BX      lr
swap    MOV     r2,r0 ; r2=a
        MOV     r0,r1 ;a=b (r0=b)
        STR     lr,[sp,#-4]! ; save lr
        LDR     r1,|L1.72|
        STR     r2,[r1,#0] ; tmp=a
        MOV     r1,r2       ; r1=tmp
        BL      swap2       ; call swap2
        MOV     r0,#0xa     ; ret value
        LDR     pc,[sp],#4 ; restore lr
main    STR     lr,[sp,#-4]!
        MOV     r0,#3  ; set up params
        MOV     r1,#4  ; before call
        BL      swap   ; to swap
        MOV     r0,#0
        LDR     pc,[sp],#4
|L1.72|
        DCD     ||.bss$2||
        AREA ||.bss||, NOINIT, ALIGN=2
tmp
```


Notes on Example 6

STR lr,[sp,#-4]!

- Pushes lr to the stack
- done in main() and swap()
- swap2() doesn't need to do it since it is a leaf call i.e. doesn't call another function
- swap() returns value in r0

Example 7: Optimizing across Functions

```
int tmp;
int swap(int a,int b);
void swap2(int a,int b);
int main(){
   int a, b, c;
   a = 3;
   b = 4;
   c = swap(a,b);
} /* end main() */
int swap(int a,int b){
   tmp = a;
   a = b;
   b = tmp;
   swap2(a,b);
} /* end swap() */
void swap2(int a,int b){
   tmp = a;
   a = b;
   b = tmp;
} /* end swap() */
```

```
        AREA ||.text||, CODE, READONLY
swap2   LDR     r1,|L1.60|
        STR     r0,[r1,#0]  ; tmp
        BX      lr
swap    MOV     r2,r0
        MOV     r0,r1
        LDR     r1,|L1.60|
        STR     r2,[r1,#0]  ; tmp
        MOV     r1,r2
        B       swap2 ; *NOT* "BL"
main PROC
        STR     lr,[sp,#-4]!
        MOV     r0,#3
        MOV     r1,#4
        BL      swap
        MOV     r0,#0
        LDR     pc,[sp],#4
|L1.60|
        DCD     ||.bss$2||
        AREA ||.bss||, tmp
||.bss$2||
        % 4
```

Doesn't return to `swap()`, instead it jumps directly back to `main()`

Compare with Example 6 - in this example, the compiler optimizes the code so that `swap2()` returns directly to `main()`

Notes on Example 7

- B swap2 **; *NOT* "BL"**
- In swap(), the tail call to swap2() is optimized away