

## Lecture 2

# Interfacing ARM Assembly Language and C



ECE349 - Spring 1998

1

## Example 1

<pre>int a,b; int main() {   a = 3;   b = 4; } /* end main() */</pre>	<pre>AREA  C\$\$code , CODE, READONLY  x\$codeseg  main   MOV     a2,#3   LDR     a1,[pc, #L00001c-.-8]   STR     a2,[a1,#0]   MOV     a2,#4   STR     a2,[a1,#4]!   MOV     a1,#0   MOV     pc,lr  L00001c DCD      x\$databeg  AREA  C\$\$data ,DATA  x\$databeg  a DCD     00000000 b DCD     00000000  EXPORT main EXPORT b EXPORT a END</pre>
---	--

label "L0001c" - compiler tends to make the labels equal to the address

pc relative (-8 'cause pc is 8 ahead of this instruction)

declare one or more words

loader will pu the address of |x\$databeg| into this memory location

declares storage (1 word) and initializes it with zero

ECE349 - Spring 1998

2

## Example 1 (con't)

Address	Code
0x00000000	AREA  C\$\$code , CODE, READONLY
0x00000004	x\$codeseq
0x00000008	main
0x0000000C	MOV a2,#3
0x00000010	LDR a1,[pc, #L00001c--8]
0x00000014	STR a2,[a1,#0]
0x00000018	MOV a2,#4
0x0000001C	STR a2,[a1,#4]!
	MOV a1,#0
	MOV pc,lr
	L00001c DCD 0x00000020 ; was originally the label  x\$dataseq
	AREA  C\$\$data ,DATA
	x\$dataseq
0x00000020	a DCD 00000000
0x00000024	b DCD 00000000
	EXPORT main
	EXPORT b
	EXPORT a
	END

*This is a pointer to the |x\$dataseq| area*

## Example 2

<pre>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() */</pre>	<pre>AREA  C\$\$code , CODE, READONLY main     STMDB sp!,{lr}     MOV a1,#3     MOV a2,#4     BL swap     MOV a1,#0     LDMIA sp!,{pc}  swap     LDR a2,[pc, #L000024--8]     STR a1,[a2,#0]     MOV pc,lr  L000024     DCD  x\$dataseq  AREA  C\$\$data ,DATA  x\$dataseq  tmp     DCD 00000000 END</pre> <p><i>STMDB - store multiple, decrement before</i>  <i>sp &lt;- sp - 4</i>  <i>mem[sp] = lr ; link register</i></p> <p><i>sp</i> → <table border="1"><tr><td>contents of lr</td></tr><tr><td> </td></tr><tr><td> </td></tr><tr><td> </td></tr></table></p>	contents of lr			
contents of lr					

## Example 3

<pre> int tmp; int *pa, *pb;  void swap(int a, int b);  int main() {     int a,b;      pa = &amp;a;     pb = &amp;b;     *pa = 3;     *pb = 4;     swap(*pa, *pb); } /* end main() */  void swap(int a,int b) {     tmp = a;     a = b;     b = tmp; } /* end swap() */ </pre>	<pre> main    STMDB    sp!,{lr}         SUB     sp,sp,#8         ADD     a2,sp,#4         LDR     a1,[pc, #L00003c--.8]         STMIB   a1!,{a2,sp}         MOV     a1,#3         STR     a1,[sp,#4]         MOV     a1,#4         STR     a1,[sp,#0]         MOV     a2,a1         LDR     a1,[sp,#4]         BL      swap         MOV     a1,#0         ADD     sp,sp,#8         LDMIA   sp!,{pc} L00003c DCD      x\$dataseg  swap    LDR     a2,[pc, #L00003c--.8]         STR     a1,[a2,#0]         MOV     pc,lr         AREA    C\$\$data ,DATA  x\$dataseg  tmp     DCD     00000000 pa      DCD     00000000 pb      DCD     00000000 </pre>
--	--

STMIB - store multiple, increment before  
sp <-- sp - 4  
mem[sp] = lr ; link register

## Example 3 (con't)

<pre> main    STMDB    sp!,{lr}         SUB     sp,sp,#8         ADD     a2,sp,#4         LDR     a1,[pc, #L00003c--.8]         STMIB   a1!,{a2,sp}         MOV     a1,#3         STR     a1,[sp,#4]         MOV     a1,#4         STR     a1,[sp,#0]         MOV     a2,a1         LDR     a1,[sp,#4]         BL      swap         MOV     a1,#0         ADD     sp,sp,#8         LDMIA   sp!,{pc} L00003c DCD      x\$dataseg  swap    LDR     a2,[pc, #L00003c--.8]         STR     a1,[a2,#0]         MOV     pc,lr         AREA    C\$\$data ,DATA  x\$dataseg  tmp     DCD     00000000 pa      DCD     00000000 pb      DCD     00000000 </pre>	<p>Stack diagram 1: Initial stack state. The stack grows downwards from higher addresses to lower addresses. Address 0x90 is the top of the stack. Address 0x80 is the bottom. The stack contains 'contents of lr' at 0x90 and 0x8C. The stack pointer (sp) is at 0x88.</p>
<p>main's local variables, a &amp; b, are placed on the stack</p>	<p>Stack diagram 2: Stack state after main's local variables are pushed. The stack pointer (sp) has moved to 0x84. The stack now contains 'contents of lr' at 0x90 and 0x8C, and local variables 'a' at 0x84 and 'b' at 0x80.</p>

## Example 3 (con't)

```

main  STMDB   sp!,{lr}
      SUB    sp,sp,#8
      ADD    a2,sp,#4
      LDR    a1,[pc, #L00003c-.-8]
      STMIB  a1!,{a2,sp}
      MOV    a1,#3
      STR    a1,[sp,#4]
      MOV    a1,#4
      STR    a1,[sp,#0]
      MOV    a2,a1
      LDR    a1,[sp,#4]
      BL    swap
      MOV    a1,#0
      ADD    sp,sp,#8
      LDMIA  sp!,{pc}
L00003c DCD    |x$databseg|
swap   LDR    a2,[pc, #L00003c-.-8]
      STR    a1,[a2,#0]
      MOV    pc,lr
      AREA  |C$$data|,DATA
|x$databseg|
tmp    DCD    00000000
pa     DCD    00000000
pb     DCD    00000000

```

loads a1 with a pointer to the start of the data segment  
stores the addresses of a and b (&a, &b) into the global variables \*pa and \*pb (which are found in the x\$databseg)

## Example 4

```

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() */

```

```

main
MOV    a3,#4
LDR    a1,[pc, #L000030-.-8]
LDR    a2,[a1,#0]
STR    a3,[a2,#0]
MOV    a2,#&a
LDR    a3,[a1,#0]
STR    a2,[a3,#4]!
MOV    a2,#&41
LDR    a1,[a1,#0]
STRB  a2,[a1,#8]
MOV    a1,#0
MOV    pc,lr
L000030 DCD    |x$databseg|
      AREA  |C$$data|,DATA
|x$databseg|
ptest  DCD    00000000

```

a1 <-- M[ #L000030] which is the pointer ptest

a2 <-- p->a

p->b = 10

watch out, ptest is only a ptr the structure was never malloc'd

## Example 5

```

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 = 4;
    d = 5;
    e = 6;
    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() */

```

```

main
    STMDB sp!,{lr}
    MOV a1,#3
    MOV a2,#4
    MOV a3,#4
    MOV ip,#5
    MOV a4,#6
    STR a4,[sp,#-4]!
    MOV a4,sp
    STMDB sp!,{a4}
    MOV a4,ip
    BL test
    MOV a1,#0
    ADD sp,sp,#8
    LDMIA sp!,{pc}
test LDR a2,[sp,#0]
    LDR a3,[pc,#L00004c--8]
    STR a1,[a3,#0]
    STR a4,[a2,#0]
    MOV pc,lr
L00004c DCD |x$dataseg|
        AREA |C$$data|,DATA
|x$dataseg|
tmp DCD 00000000

```

parameters a, b, c & d  
are stored in regs

overflow of test()  
parameters - pushes  
parms onto the stack

save the stack pointer --  
Oa4 <-- sp0 so that the  
preincrement won't affect the  
address save on the stack

a1 holds the return value







## Passing and Returning Structures

---

- u Structures are usually passed in registers (and overflow onto the stack when necessary)
- u When a function returns a struct, a pointer to where the struct result is to be placed is passed in a1 (first parameter)
  - Example

```
struct s f(int x);  
-- is compiled as --  
void f(struct s *result, int x);
```

## Example - Passing Structures as Parameters

---

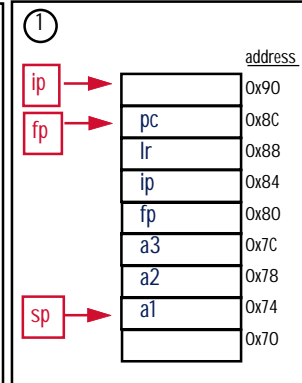
```
typedef struct two_ch_struct{  
    char ch1;  
    char ch2;  
} two_ch;  
  
two_ch max(two_ch a, two_ch b){  
    return((a.ch1 > b.ch1) ? a : b);  
} /* end max() */
```

```
max  
MOV    ip, sp  
STMDB sp!, {a1-a3, fp, ip, lr, pc}  
SUB    fp, ip, #4  
LDRB  a3, [fp, #-&14]  
LDRB  a2, [fp, #-&10]  
CMP   a3, a2  
SUBLE a2, fp, #&10  
SUBGT a2, fp, #&14  
LDR   a2, [a2, #0]  
STR   a2, [a1, #0]  
LDMDB fp, {fp, sp, pc}
```

## Example (con't)

```

max
MOV    ip, sp
STMDB sp!, {a1-a3, fp, ip, lr, pc}
SUB    fp, ip, #4
LDRB  a3, [fp, #-&14] ; a3 <-- a2 from stack
LDRB  a2, [fp, #-&10] ; a2 <-- a3 from stack
CMP    a3, a2
SUBLE  a2, fp, #&10 ; a2 <-- pointer to max
SUBGT  a2, fp, #&14
LDR    a2, [a2, #0] ; grab the max value
STR    a2, [a1, #0] ; store it into a1
LDMDB  fp, {fp, sp, pc}
    
```



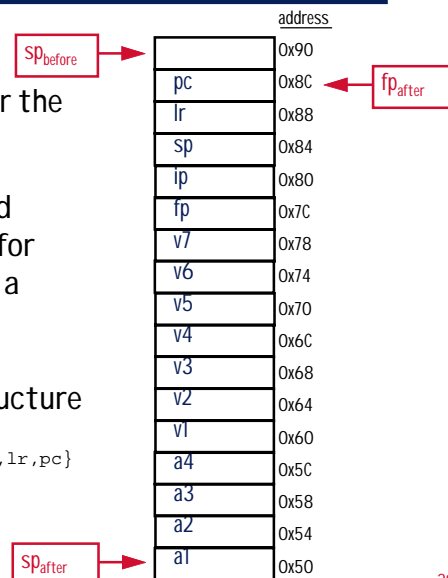
• frame pointer (fp) points to top of stack for function

## Frame Pointer

- u Points to top of stack area for the current function
  - Or zero if not being used
- u By using the frame pointer and storing it at the same offset for every function call, it creates a singly-linked list of **activation records**
- u Creating stack backtrace structure

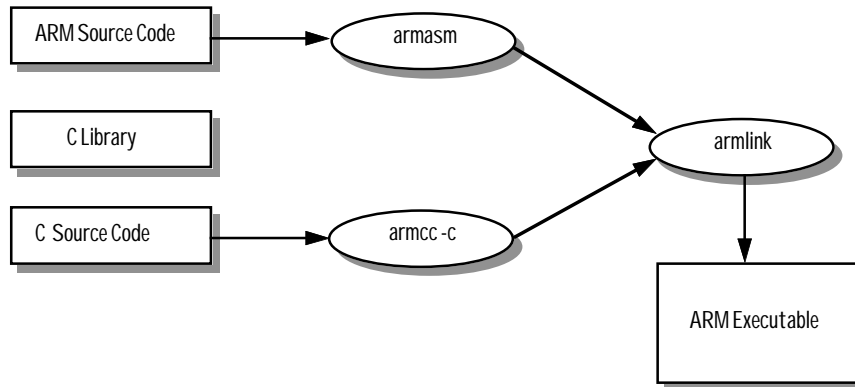
```

MOV    ip, sp
STMFD  sp!, {a1-a4, v1-v5, sb, fp, ip, lr, pc}
SUB    fp, ip, #4
    
```



## Mixing C and Assembly Language

---



## ARM C Library

---

- u Two libraries
  - ANSI C library
  - Minimal standalone library
    - 1 division and remainder functions
    - 1 stack-limit checking functions
    - 1 lowest-level memory management
    - 1 program startup (calling main())
    - 1 program termination (\_exit())
  - We use the ANSI C standard library for now
    - 1 /afs/ece/class/ee349/VLSI/lib/armlib.32l <-- l is for little endian

## Running armsd

---

- u armsd is a software simulator of the ARM architecture
- u /afs/ece/class/ee349/bin/armsd
  - armsd fileName
  - load fileName
- u Ignore the documentation on the armsd - the docs explain how to extend the simulator - not how to use it
- u The simulator has the debugger built into it - so you need to learn how to use the debugger
- u Lots of commands:
  - See Jumpstart Programming Techniques - Chapter 1 (Getting Started) and VLSI Reference Manual Chapter 7 - Symbolic Debugger
  - Takes a bit of time to learn how to use

## Multiply

---

- u Multiply instruction can take multiple cycles
- u Can convert  $Y * \text{Constant}$  into series of adds and shifts
  - $Y * 9 = Y * 8 + Y * 1$
  - Assume R1 holds Y and R2 will hold the result

```
RSB  R2, R1, R1, LSL #3 ; LSL #3 is the same as multiply by 8
ADD  R2, R2, R1
```
  - Another example:  $Y * 105$ 

```
105  = 128 - 23
      = 128 - (16 + 7)
      = 128 - (16 + (8 - 1))

RSB  r2, r1, r1, LSL #3 ; r2 <-- Y * 7 = Y * 8 - Y (assume r1 holds Y)
ADD  r2, r2, r1, LSL #4 ; r2 <-- r2 + Y * 4
RSB  r2, r2, r1, LSL #7 ; r2 <-- (Y * 128) - r2
```
  - Or  $Y * 105 = Y * (15 * 7) = Y * (16 - 1) * (8 - 1)$ 

```
RSB  r2, r1, r1, LSL #4 ; r2 <-- (r1 * 16) - r1 is TMP = (Y * 16) - (Y * 1)
RSB  r2, r2, r2, LSL #3 ; r2 <-- (r2 * 8) - r2 is (TMP * 8) - (TMP - 1)
```