Programming the ARM

Computer Design 2002, Lecture 4

Robert Mullins

Quick Recap

- The Control Flow Model
  - Ordered list of instructions, fetch/execute, PC
- Instruction Set Architectures
  - Types of internal storage
    - Accumulator, stack and general purpose register machines
  - Addressing modes
    - Register indirect, displacement, memory indirect *etc.*
  - Branch conditions
    - Condition codes, condition registers, branch and compare
  - Encoding Instructions
    - Fixed length or variable length encodings
    - Representing immediates, opcode and operands

Quick Recap

- Making the common case fast and Amdahl's Law

  - See H&P (chapters 1 and 2)

- The Big Picture - Interaction between the compiler, architecture and instruction set
  - E.g. a poorly designed instruction set may make high-performance implementation difficult and restrict the effectiveness of an optimizing compiler.

Lecture 4

- This Lecture
  - Implementing functions and procedures
  - The ARM Procedure Call Standard (APCS)
  - Development tools
  - Practical 4 of the ECAD labs

Functions and Procedures

- Structure program (abstraction/hierarchy)
- Package useful code so it can be reused
  - Must use a well defined interface
- Questions
  - How do we pass arguments to the function?
  - How do we obtain the result?

Procedure Calls

- Register values are preserved by saving and restoring them from a stack in memory
- Agree on a standard, define how registers are used (part of ABI – Application Binary Interface)
  - Which registers hold arguments
  - Which register holds the result (or pointer to)
  - Some registers may need to be saved (preserved) by caller (**caller-saved**)
  - Some may need to be saved by callee (**callee-saved**)

ARM Procedure Call Standard

- There are in fact many variants, lets look at base standard
- Four argument registers (r0-r3)
  - Not preserved by routine/function (callee)
  - May be saved by caller if necessary
  - May be used to return results to caller
- Registers r4-r11 are typically used to hold the routine's local variables
  - The value of these registers remains unchanged after a subroutine call
  - Registers may need to be saved and restored by callee
- Registers r12-r15 have special dedicated roles
  - e.g. the link register holds the return address

APCS Register Usage Convention

Register	APCS Name	APCS Role
0	a1	argument 1 / result / scratch register for function
1	a2	argument 2 / result / scratch register for function
2	a3	argument 3 / result / scratch register for function
3	a4	argument 4 / result / scratch register for function
4	v1	variable register 1
5	v2	variable register 2
6	v3	variable register 3
7	v4	variable register 4
8	v5	variable register 5
9	sb or v6	static base or variable register 6
10	sl or v7	stack limit or variable register 7
11	fp	frame pointer (pointer to start of current stack frame) or variable register 8
12	ip	the intra-procedure-call scratch register
13	sp	lower end of current stack frame (stack pointer)
14	lr	the link register (return address)
15	pc	the program counter

Parameter Passing and result return

- a1-a4 (r0-r3) may be used to hold parameters
- Additional parameters may be passed on the stack
- Results may be returned in a1-a4 as a value or indirectly as an address

Simple function call

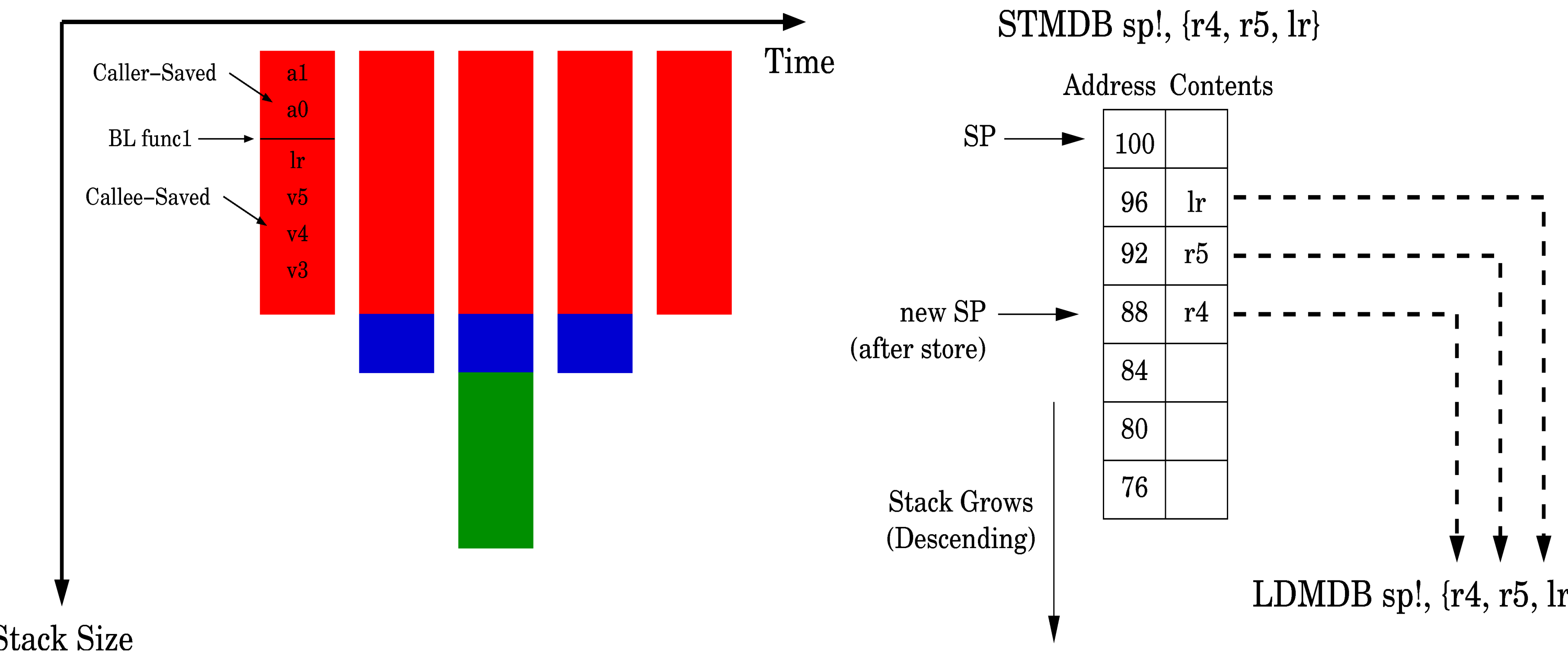
```
        MOV     r0, #10
        MOV     r1, #5
        BL      max     ;
        …..
max     CMP     r0, r1
        MOVLT   r0, r1  ; if r0<r1 r0=r1
        MOV     pc, lr
```

For simple leaf functions that only use r0-r3 (a1-a4) the function overhead is small as no registers need to be saved. In real programs around half the functions may be simple leaf functions like this.

Saving and restoring registers

```
            BL      myfunction
            …..
myfunction  STMFD sp!, {r4-r10, lr}     ; save registers
            …..
            …..
            LDMFD sp!, {r4-r10, pc}     ; restore and
                                        ; return
```

The Stack



Full-descending stack - ARM stack grows down, stack pointer points to last entry (not next free entry)

ARM Assembler Examples

Assembler Directives

- Information for the assembler
- Common uses
  - Define and name new section, code or data
  - Constants, aliases
  - Allocate bytes of memory as data and initialize contents

Allocating memory for data

- Allocate bytes of memory (DCB)

```
C_String    DCB    "MyString", 0
```

- Allocate words of memory (DCD)

```
Data        DCD    1234, 1, 5, 20
```

- Reserve a 'zeroed' block of memory (SPACE or %)

```
table    % 1024    ; 1024 bytes of zeroed
                   ; memory
```

Writing Assembler, Hello World Example

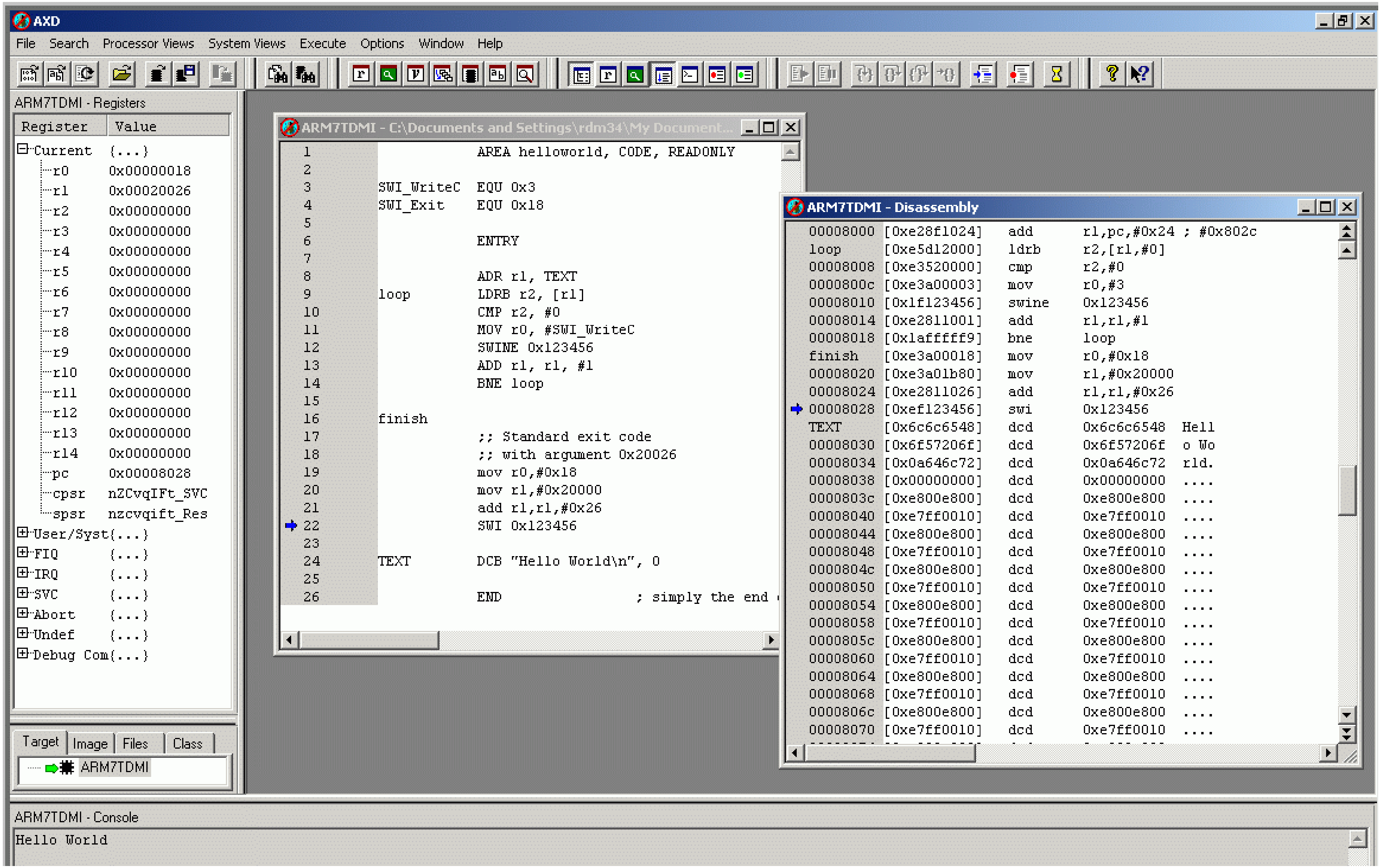
```
                ; create new code area called 'hello'
                AREA    hello, CODE, READONLY

SWI_WriteC      EQU     0x3                 ; symbolic name for constant
SWI_Exit        EQU     0x18
                ENTRY                       ; entry point into program

START           ADR     r1, TEXT            ; ADD r1, pc, offset of TEXT
LOOP            LDRB    r2, [r1]            ; load character
                CMP     r2, #0              ; check we are not at end of string
                MOV     r0, #SWI_WriteC     ; call putchar (r1 points to char)
                SWINE   0x123456            ; do system call
                ADD     r1, r1, #1          ; increase our pointer
                BNE     LOOP                ; loop if we are not finished

                MOV     r0, #SWI_Exit       ; standard exit system call
                MOV     r1, #0x20000
                ADD     r1,r1,#0x26
                SWI     0x123456

TEXT            DCB     "Hello World\n", 0
                END                         ; end of source file!
```


AXD
File Search Processor Views System Views Execute Options Window Help
ARM7TDMI - Registers
Register Value
Current {...}
r0 0x00000018
r1 0x00020026
r2 0x00000000
r3 0x00000000
r4 0x00000000
r5 0x00000000
r6 0x00000000
r7 0x00000000
r8 0x00000000
r9 0x00000000
r10 0x00000000
r11 0x00000000
r12 0x00000000
r13 0x00000000
r14 0x00000000
pc 0x00008028
cpsr nZCvqIFt_SVC
spsr nzcvqift_Res
User/Syst{...}
FIQ {...}
IRQ {...}
SVC {...}
Abort {...}
Undef {...}
Debug Com{...}
Target Image Files Class
ARM7TDMI
ARM7TDMI - C:\Documents and Settings\rdm34\My Document...
1 AREA helloworld, CODE, READONLY
2
3 SWI_WriteC EQU 0x3
4 SWI_Exit EQU 0x18
5
6 ENTRY
7
8 ADR r1, TEXT
9 loop LDRB r2, [r1]
10 CMP r2, #0
11 MOV r0, #SWI_WriteC
12 SWINE 0x123456
13 ADD r1, r1, #1
14 BNE loop
15
16 finish
17 ;; Standard exit code
18 ;; with argument 0x20026
19 mov r0,#0x18
20 mov r1,#0x20000
21 add r1,r1,#0x26
22 SWI 0x123456
23
24 TEXT DCB "Hello World\n", 0
25
26 END ; simply the end
ARM7TDMI - Disassembly
00008000 [0xe28f1024] add r1,pc,#0x24 ; #0x802c
loop [0xe5d12000] ldrb r2,[r1,#0]
00008008 [0xe3520000] cmp r2,#0
0000800c [0xe3a00003] mov r0,#3
00008010 [0x1f123456] swine 0x123456
00008014 [0xe2811001] add r1,r1,#1
00008018 [0x1afffff9] bne loop
finish [0xe3a00018] mov r0,#0x18
00008020 [0xe3a01b80] mov r1,#0x20000
00008024 [0xe2811026] add r1,r1,#0x26
00008028 [0xef123456] swi 0x123456
TEXT [0x6c6c6548] dcd 0x6c6c6548 Hell
00008030 [0x6f57206f] dcd 0x6f57206f o Wo
00008034 [0x0a646c72] dcd 0x0a646c72 rld.
00008038 [0x00000000] dcd 0x00000000 ....
0000803c [0xe800e800] dcd 0xe800e800 ....
00008040 [0xe7ff0010] dcd 0xe7ff0010 ....
00008044 [0xe800e800] dcd 0xe800e800 ....
00008048 [0xe7ff0010] dcd 0xe7ff0010 ....
0000804c [0xe800e800] dcd 0xe800e800 ....
00008050 [0xe7ff0010] dcd 0xe7ff0010 ....
00008054 [0xe800e800] dcd 0xe800e800 ....
00008058 [0xe7ff0010] dcd 0xe7ff0010 ....
0000805c [0xe800e800] dcd 0xe800e800 ....
00008060 [0xe7ff0010] dcd 0xe7ff0010 ....
00008064 [0xe800e800] dcd 0xe800e800 ....
00008068 [0xe7ff0010] dcd 0xe7ff0010 ....
0000806c [0xe800e800] dcd 0xe800e800 ....
00008070 [0xe7ff0010] dcd 0xe7ff0010 ....
ARM7TDMI - Console
Hello World

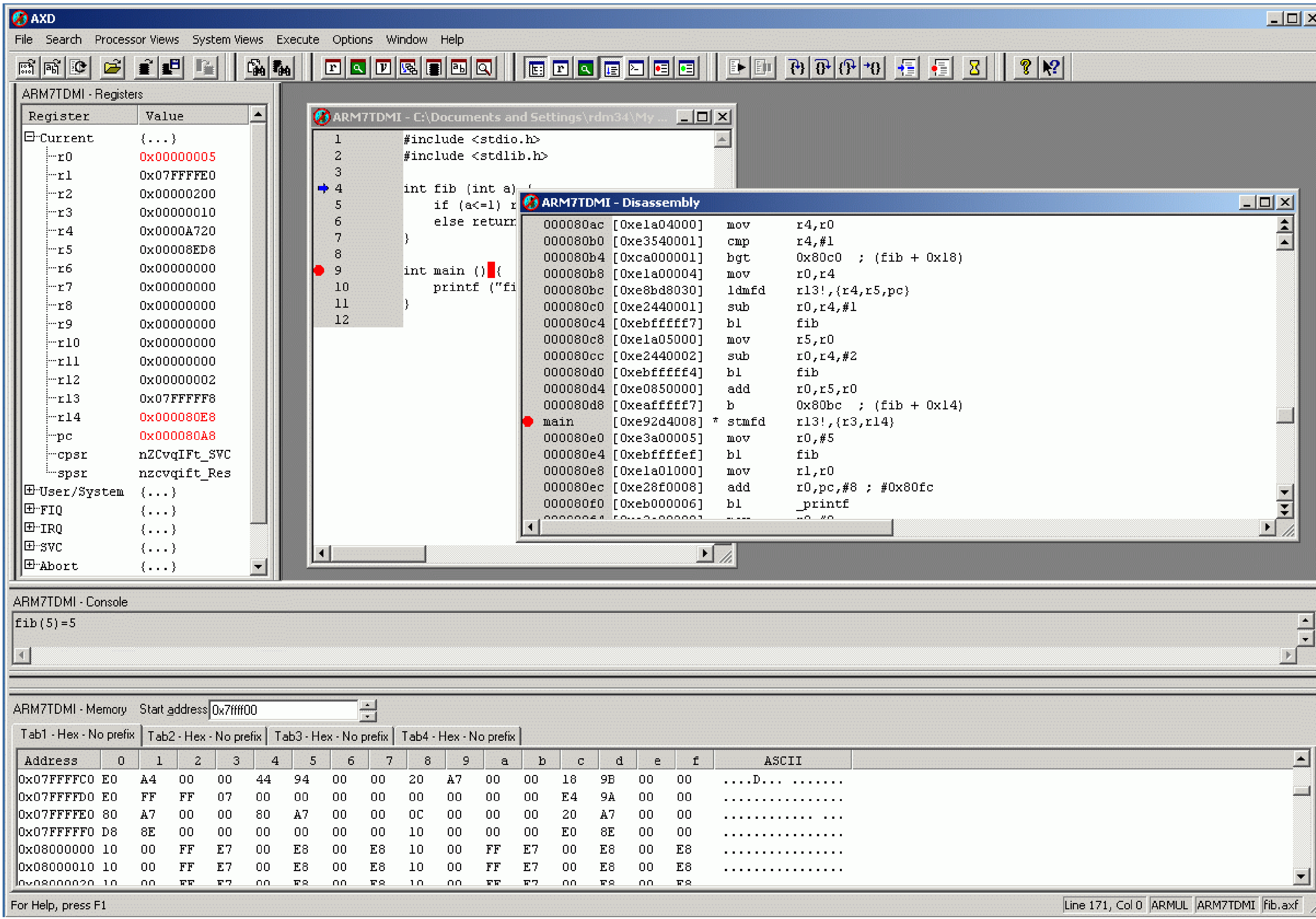
The program in memory….

ARM7TDMI - Disassembly

```
  00008000 [0xe28f1024]   add    r1,pc,#0x24 ; #0x802c
  loop     [0xe5d12000]   ldrb   r2,[r1,#0]
  00008008 [0xe3520000]   cmp    r2,#0
  0000800c [0xe3a00003]   mov    r0,#3
  00008010 [0x1f123456]   swine  0x123456
  00008014 [0xe2811001]   add    r1,r1,#1
  00008018 [0x1afffff9]   bne    loop
  finish   [0xe3a00018]   mov    r0,#0x18
  00008020 [0xe3a01b80]   mov    r1,#0x20000
  00008024 [0xe2811026]   add    r1,r1,#0x26
➡ 00008028 [0xef123456]   swi    0x123456
  TEXT     [0x6c6c6548]   dcd    0x6c6c6548  Hell
  00008030 [0x6f57206f]   dcd    0x6f57206f  o Wo
  00008034 [0x0a646c72]   dcd    0x0a646c72  rld.
  00008038 [0x00000000]   dcd    0x00000000
```

```
int fib (int a) {
        if (a<=1) return a;
        else return fib (a-1)+fib(a-2);
    }

fib:
    stmfd     r13!, {r4, r5, r14}     ; preserve registers
    mov       r4, r0                  ; r4=a
    cmp       r0, #1                  ; compare a with 1
    movle     r0, r4                  ; (if a<=1) result=a
    ldmlefd   r13!, {r4, r5, pc}      ; (if a<=1) return
    sub       r0, r4, #1
    bl        fib                     ; fib (a-1)
    mov       r5, r0                  ; r5=fib(a-1)
    sub       r0, r4, #2
    bl        fib                     ; fib (a-2)
    add       r0, r5, r0              ; result=fib(a-1)+fib(a-2)
    ldmfd     r13!, {r4, r5, pc}      ; return
```

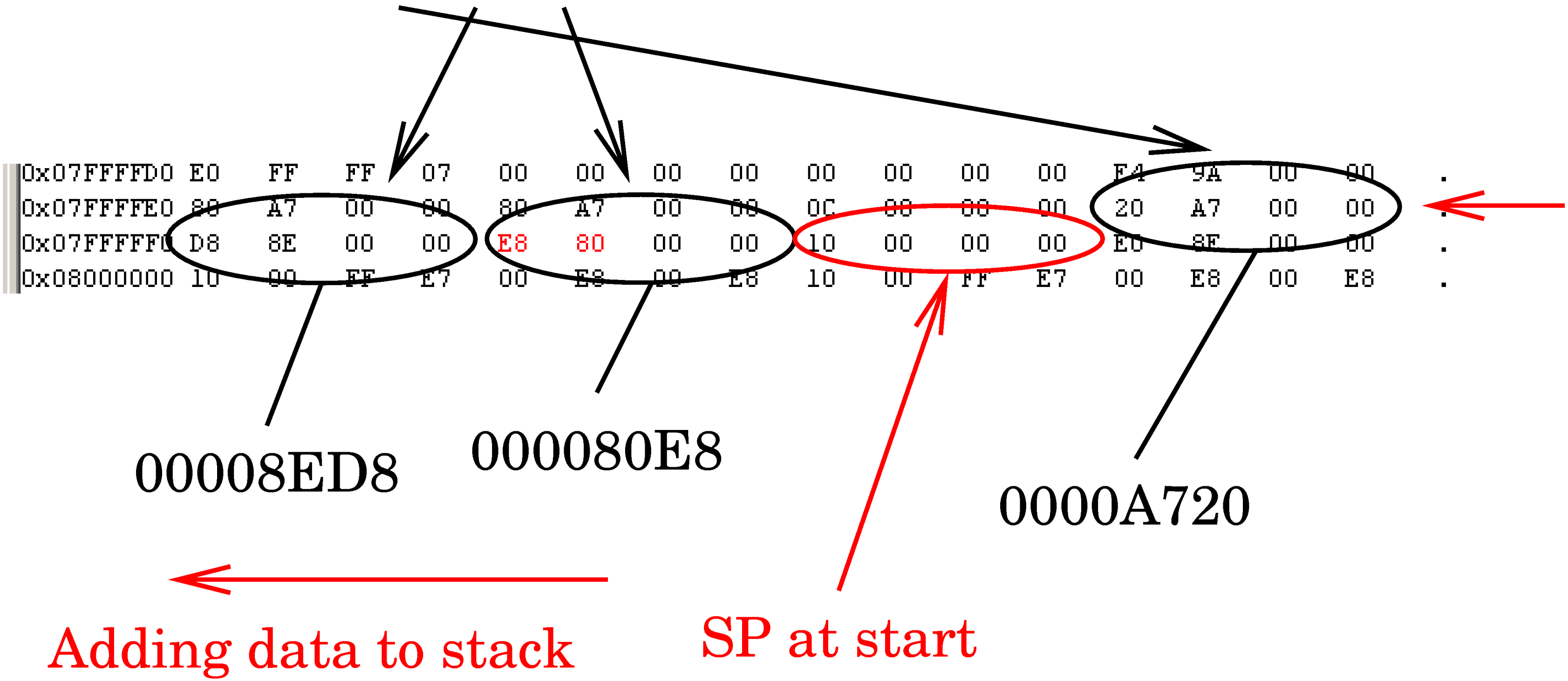

AXD
File Search Processor Views System Views Execute Options Window Help
ARM7TDMI - Registers
Register Value
Current {...}
r0 0x00000005
r1 0x07FFFFE0
r2 0x00000200
r3 0x00000010
r4 0x0000A720
r5 0x00008ED8
r6 0x00000000
r7 0x00000000
r8 0x00000000
r9 0x00000000
r10 0x00000000
r11 0x00000000
r12 0x00000002
r13 0x07FFFFF8
r14 0x000080E8
pc 0x000080A8
cpsr nZCvqIFt_SVC
spsr nzcvqift_Res
User/System {...}
FIQ {...}
IRQ {...}
SVC {...}
Abort {...}
ARM7TDMI - C:\Documents and Settings\rdm34\My ...
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int fib (int a) {
5 if (a<=1) r
6 else return
7 }
8
9 int main () {
10 printf ("fi
11 }
12
ARM7TDMI - Disassembly
000080ac [0xe1a04000] mov r4,r0
000080b0 [0xe3540001] cmp r4,#1
000080b4 [0xca000001] bgt 0x80c0 ; (fib + 0x18)
000080b8 [0xe1a00004] mov r0,r4
000080bc [0xe8bd8030] ldmfd r13!,{r4,r5,pc}
000080c0 [0xe2440001] sub r0,r4,#1
000080c4 [0xebfffff7] bl fib
000080c8 [0xe1a05000] mov r5,r0
000080cc [0xe2440002] sub r0,r4,#2
000080d0 [0xebfffff4] bl fib
000080d4 [0xe0850000] add r0,r5,r0
000080d8 [0xeafffff7] b 0x80bc ; (fib + 0x14)
main [0xe92d4008] * stmfd r13!,{r3,r14}
000080e0 [0xe3a00005] mov r0,#5
000080e4 [0xebffffef] bl fib
000080e8 [0xe1a01000] mov r1,r0
000080ec [0xe28f0008] add r0,pc,#8 ; #0x80fc
000080f0 [0xeb000006] bl _printf
ARM7TDMI - Console
fib(5)=5
ARM7TDMI - Memory Start address 0x7ffff00
Tab1 - Hex - No prefix Tab2 - Hex - No prefix Tab3 - Hex - No prefix Tab4 - Hex - No prefix
Address 0 1 2 3 4 5 6 7 8 9 a b c d e f ASCII
0x07FFFFC0 E0 A4 00 00 44 94 00 00 20 A7 00 00 18 9B 00 00 ....D... ....
0x07FFFFD0 E0 FF FF 07 00 00 00 00 00 00 00 00 E4 9A 00 00 ....
0x07FFFFE0 80 A7 00 00 80 A7 00 00 0C 00 00 00 20 A7 00 00 ....
0x07FFFFF0 D8 8E 00 00 00 00 00 00 10 00 00 00 E0 8E 00 00 ....
0x08000000 10 00 FF E7 00 E8 00 E8 10 00 FF E7 00 E8 00 E8 ....
0x08000010 10 00 FF E7 00 E8 00 E8 10 00 FF E7 00 E8 00 E8 ....
For Help, press F1
Line 171, Col 0 ARMUL ARM7TDMI fib.axf

R13 = 0x07FFFFF8 (stack pointer)

R4 = A720, R5 = 8ED8

R14 = 80E8 (address after BL fib instruction in main)

STMFD r13!, {r4, r5, r14}



Fib(5)

R5

R4

Return Address to Main()

R4=5

Fib(R4-1)

R5

R4 (a=5)

Return address

Fib(3)….. *etc.*

Fib(R4-2)

Return sum

ECAD Workshop Four

Sieve of Eratosthenes

2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0

2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	1	0	1	0	1	0	1	0

2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	1	1	1	0	1	0	1	1

Later Workshops

- Make sure you use procedure call standard correctly
  - Use branch-and-link (BL) instruction to call procedure
  - Pass parameters in correct registers
  - Save registers when necessary
  - Restore registers on exit

Next Lecture

- OS Support and Memory Management
  - Virtual Memory
  - Interrupts/Exceptions
  - ARM specifics (operating modes, page table organisation etc.)

Self Study/Supervision Work

- Write a simple C program to sum the numbers 1 to n, compile it and examine the assembler produced
  - You might want to try executing the program in the debugging environment, this will allow you to single step through the program. You can look at the contents of the registers and memory as you go.
- Challenge
  - Write a program (in ARM assembly language) to reverse the bytes of a 32-bit register
  - It's possible to do this using only one additional register to hold temporary results and 4 ARM instructions!