ARM Memory Addressing and Function Calls

Tom Kelliher, CS 220

1 Administrivia

Today's Objectives

1. Use indirect addressing to move data between registers and memory.
2. Manipulate numeric and character arrays.
3. Use subroutine call and return instructions to implement functions without a function call stack.

Next Up

Read 3.9

1. Study several examples of ARM instruction sequences and write a few ARM programs.

2 Warm-Up

1. The ARM instruction

```
ldr r0, [r1]
```

   (a) Loads the contents of r1 into r0.

   (b) Loads the contents of the memory word who's address is contained in r1 into r0.

   (c) Loads the contents of the memory byte who's address is contained in r1 into r0. The memory byte is loaded into the least significant byte (lsB) of the register, while the three upper bytes of the register are cleared.

   (d) Loads the contents of the memory byte who's address is contained in r1 into r0. The memory byte is loaded into the least significant byte (lsB) of the register, while the three upper bytes of the register are not modified.

2. The ARM instruction

```
ldrb r0, [r1]
```

   (a) Loads the contents of r1 into r0.

   (b) Loads the contents of the memory word who's address is contained in r1 into r0.

   (c) Loads the contents of the memory byte who's address is contained in r1 into r0. The memory byte is loaded into the least significant byte (lsB) of the register, while the three upper bytes of the register are cleared.

   (d) Loads the contents of the memory byte who's address is contained in r1 into r0. The memory byte is loaded into the least significant byte (lsB) of the register, while the three upper bytes of the register are not modified.

3. Consider this ARM program fragment:

```
        area prog, code, readonly
        entry
        ldr r0, =str   ; Load base address of str into r0.
        add r0, r0, #4
        ldrb r1, [r0]

        area progData, data, readonly
str     = "Hello class!\0"   ; Ensure that str is null-terminated.
        end
```

Following the execution of the `ldrb` instruction, r1 contains

(a) The ASCII character ’H’.

(b) The ASCII character ’e’.

(c) The ASCII character ’l’.

(d) The ASCII character ’o’.

(e) The ASCII NULL character.

4. Consider this ARM program fragment:

```
        area prog, code, readonly
        entry
        ldr r0, =darray   ; Load base address of darray into r0.
        add r0, r0, #4
        ldr r1, [r0]

        area progData, data, readonly
darray  dcd -10, 45, 37, 5, 9   ; Load integer values into successive
                                ; memory words.
        end
```

Following the execution of the `ldrb` instruction, r1 contains

(a) The integer value -10.

(b) The integer value 10.

(c) The integer value 45.

(d) The integer value 9.

(e) Gee, I hope Arlen brings me a tasty beverage today.

5. Consider this ARM program fragment:

```
        area prog, code, readonly
        entry
        ldr r0, =darray   ; Load base address of darray into r0.
        mov r1, #4
        ldr r2, [r0, r1, lsl #2]

        area progData, data, readonly
darray  dcd -10, 45, 37, 5, 9   ; Load integer values into successive
                               ; memory words.
        end
```

Following the execution of the `ldrb` instruction, r1 contains

(a) The integer value -10.

(b) The integer value 45.

(c) The integer value 37.

(d) The integer value 5.

(e) The integer value 9.

6. Consider this ARM program:

```
        area prog, code, readonly
        entry
        mov r1, #-9
        mov r2, #2
again   bl add
        mov r1, r0
        bl add
        mov r1, r0
        cmp r1, #0
        ble again

fin     b fin

add     add r0, r1, r2
        bx lr
        end
```

`add` is a function. r1 behaves as

(a) the function's return value.

(b) the function's parameter.

(c) the function's return address.

(d) the function's base address.

3 Problems

1. Manually compile the following C program into a working ARM program in uVision:

```
#include <stdio.h>

int data[] = { 62, 41, -19, 35, 73, 66, 12, 21, -24, 22 };

int main()
{
   int i;
   int n;
   int min;
   int max;

   /* sizeof() returns the size of an object in bytes. */

   n = sizeof(data) / sizeof(int);

   min = max = data[0];

   for (i = 1; i < n; i++)
      if (data[i] < min)
         min = data[i];
      else if ( data[i] > max)
         max = data[i];

   printf("Max: %d     Min: %d\n", max, min);
   return 0;
}
```

2. Manually compile the following C program into a working ARM program in uVision:

```
#include <stdio.h>

char str1[] = "I am a string.";
char str2[] = "I am a string joke.";

int main()
{
   char *ptr1;
   char *ptr2;
   int equal;

   ptr1 = str1;
   ptr2 = str2;

   while (*ptr1 == *ptr2 && *ptr1 != '\0')
   {
      ptr1++;
      ptr2++;
   }

   if (*ptr1 == *ptr2)
      equal = 1;
   else
      equal = 0;

   printf("%d\n", equal);
   return 0;
}
```

3. Manually compile the following C program into a working ARM program in uVision:

```
#include <stdio.h>
int fact(int n);

int main()
{
   int sum;
   int i;
   int j;

   j = fact(12);

   /* The value printed is 0x1C8CFC00. */
   printf("Hexadecimal: %X\n", j);

   sum = 0;

   for (i = 0; i < 10; i++)
      sum += fact(i);

   /* The value printed is 0x63E1A. */
   printf("Hexadecimal: %X\n", sum);
   return 0;
}

int fact(int n)
{
   int fact;
   int i;

   if (n < 0)
      fact = -1;
   else
   {
      fact = 1;

      for (i = 1; i <= n; i++)
         fact *= i;
   }

   return fact;
}
```