Computer Organization
How to access data in ARM – Addressing Modes

Prof. Charles W. Kann

Register Conventiions

r0	Argument and Result	Not Preserved
r1	Argument	Not Preserved
r2	Argument	Not Preserved
r3	Argument	Not Preserved
r4	General	Preserved
r5	General	Preserved
r6	General	Preserved
r7	General	Preserved
r8	General	Preserved
f9	General	Preserved
r10	General	Preserved
r11	General	Preserved
r12	General	Preserved
r13	General	Not Preserved
r14	General	Preserved

Why are there different ARM addressing Modes

- Before continuing with ARM assembly, it is necessary to cover how data will be accessed in an ARM assembly language program.
- Because program data is of different types and used it different ways, it will be stored in different places using different addressing. No one type of addressing is effective or efficient for all uses, so a number of different addressing mechanisms are needed.
- For example, an integer is 4 bytes and can be stored in a register when it needs to be used.
- Registers are limited in number, so an integer not being used needs to be stored in memory.
- A string is a null terminated character (byte) array of indeterminate length, and must be maintained in memory.

Load/Store Architecture

- **ARM is a load/store architecture. This means that all values must be loaded from memory to a register before being used, and stored to a register to save the values (note: Rd, Rn, Tm are just some registers, but generally Rd is a destination).**

- **The load instruction is of the form:**

  ldr Rd, [Rn, #immediate] # Store value at address Rn + #immediate into Rd

  Meaning load into Rd whatever is in Rn plus the value of the #immediate

  - This ldr can be used with a label to be "ldr Rd, =Label". The assembler will translate Label into a real address

- **The store instruction is of the form:**

  - str Rm, [Rn, #immediate] # Load value in Rd to address Rn + #immediate

  - Same rules apply as for ldr

ARM addressing Modes

- **The following addressing modes used in ARM architecture will be covered in this module.**
  - Direct – The value is stored at a know address, e.g. a lable
  - Immediate – Value is par of the instruction
  - Register direct – Value is stored in a register
  - Direct – value is stored at a memory location
  - Register Indirect – Address of variable is stored in register
  - Register indirect with offset – Address of variable is register + offset

ARM addressing Modes

- **The following addressing modes used in ARM architecture will not be covered in this module. Starred items will be covered later.**
  - Register indirect Register indexed – Address of variable is register + register *
  - Register indirect with pre-increment
  - Register indirect with post-increment
  - Register indirect with offset – Address of variable is register + offset *
  - Register indirect Register indexed – Address of variable is register + register *

Address modes in IOExample_1.s

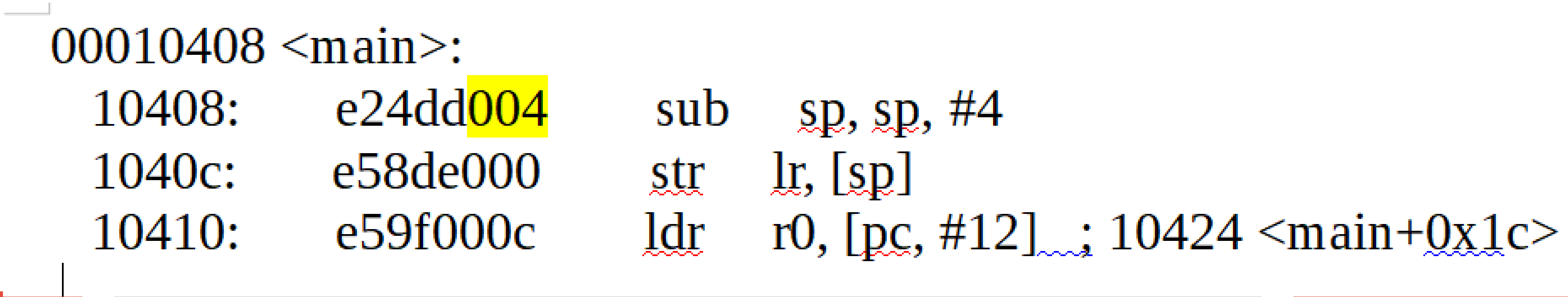
```
main:
# Save return to os on stack
   sub sp, sp, #4
   str lr, [sp, #0]

# Printing The Message
   ldr r0, =HelloWorld
   bl  printf

# Return to the OS
   ldr lr, [sp, #0]
   add sp, sp, #4
   mov pc, lr

.data

HelloWorld:
   .asciz "Hello World\n"
```

Immediate mode addressing

- The following line uses Immediate mode addressing for the value #4.

  sub sp, sp, #4

- The value 4 is part of the instruction that was translated. Note that program objdump created the following machine code output for the compiled code. . The highlighted value of 4 is the immediate value stored in the machine code instruction.



```
00010408 <main>:
   10408:      e24dd004      sub    sp, sp, #4
   1040c:      e58de000      str    lr, [sp]
   10410:      e59f000c      ldr    r0, [pc, #12]  ; 10424 <main+0x1c>
```

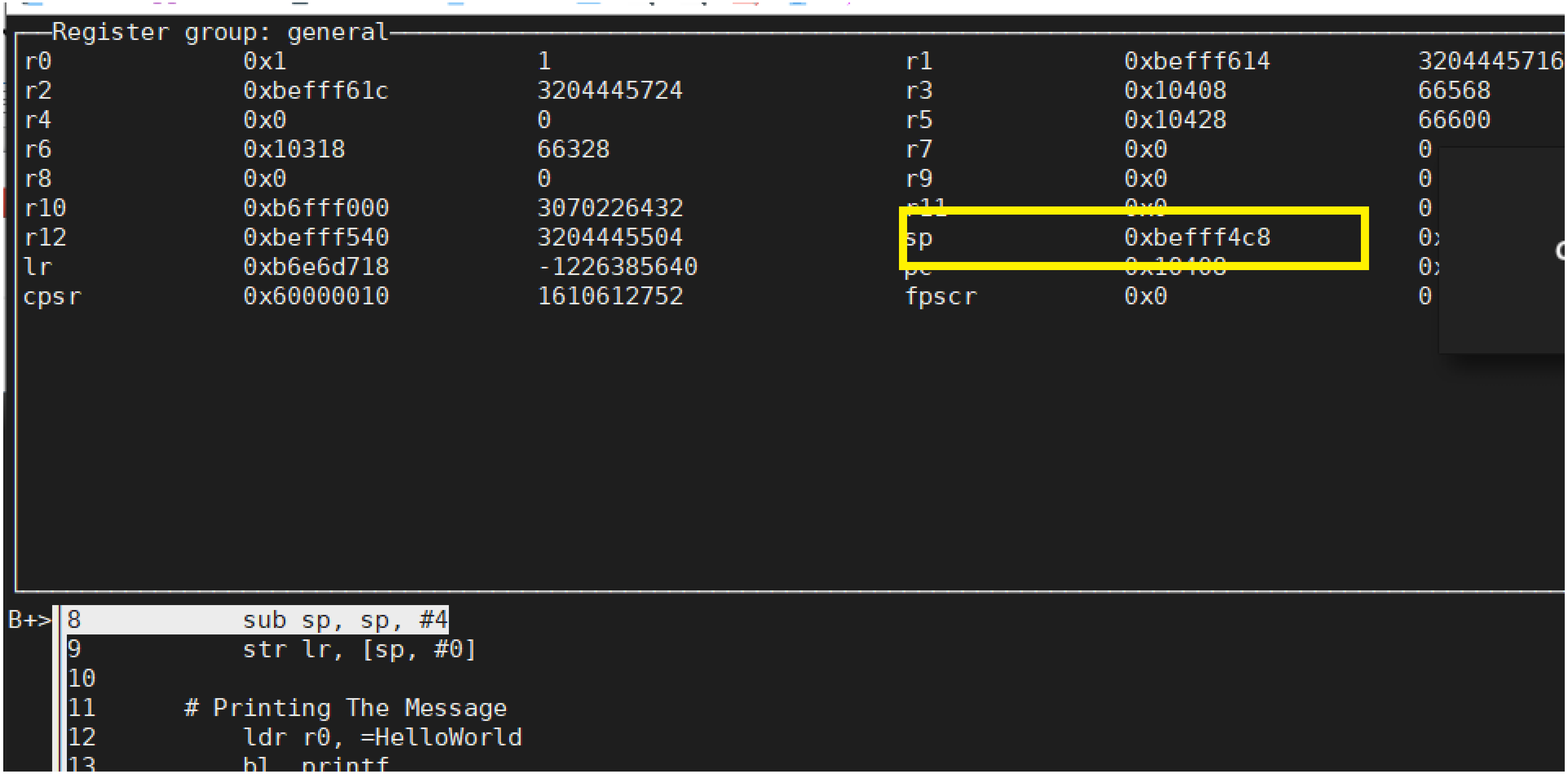
Register direct addressing

- The following line uses Immediate mode addressing for the value #4.

```
sub sp, sp, #4
```

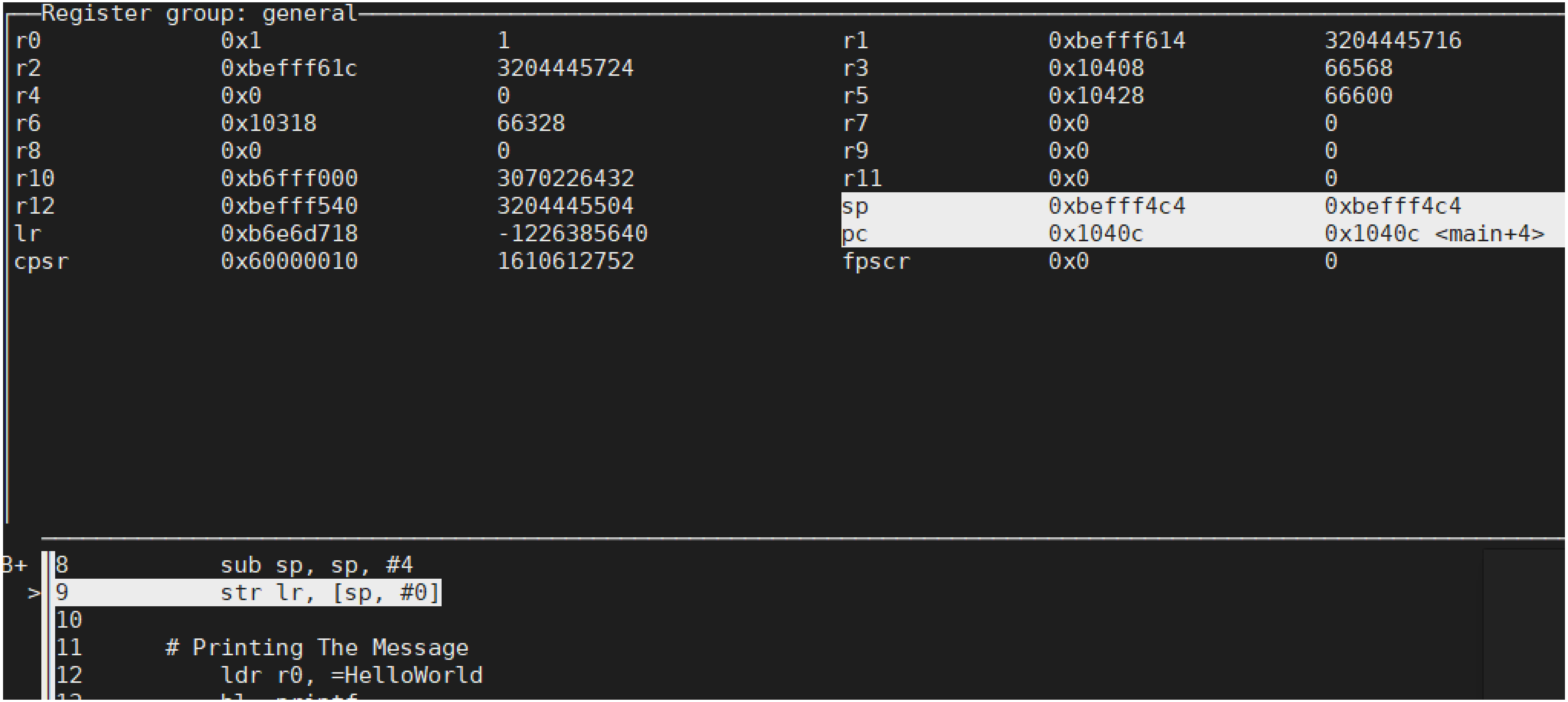
- sp is a register that contains a value
- The next two slides show (in gdb) how that value is change before and after the instruction is executed.

Before running line 8



```
Register group: general
r0             0x1                 1                        r1             0xbefff614          3204445716
r2             0xbefff61c          3204445724               r3             0x10408             66568
r4             0x0                 0                        r5             0x10428             66600
r6             0x10318             66328                    r7             0x0                 0
r8             0x0                 0                        r9             0x0                 0
r10            0xb6fff000          3070226432               r11            0x0                 0
r12            0xbefff540          3204445504               sp             0xbefff4c8          0x
lr             0xb6e6d718          -1226385640              pc             0x10408             0x
cpsr           0x60000010          1610612752               fpscr          0x0                 0

B+> 8           sub sp, sp, #4
    9           str lr, [sp, #0]
    10
    11      # Printing The Message
    12          ldr r0, =HelloWorld
    13          bl  printf
```

After running line 8



```
┌─Register group: general──────────────────────────────────────────────────────────────────────────
│r0             0x1                 1                       r1             0xbefff614          3204445716
│r2             0xbefff61c          3204445724              r3             0x10408             66568
│r4             0x0                 0                       r5             0x10428             66600
│r6             0x10318             66328                   r7             0x0                 0
│r8             0x0                 0                       r9             0x0                 0
│r10            0xb6fff000          3070226432              r11            0x0                 0
│r12            0xbefff540          3204445504              sp             0xbefff4c4          0xbefff4c4
│lr             0xb6e6d718          -1226385640             pc             0x1040c             0x1040c <main+4>
│cpsr           0x60000010          1610612752              fpscr          0x0                 0
│
│
│
│
│
│
│
   ──────────────────────────────────────────────────────────────────────────────────────────────
B+ │8            sub sp, sp, #4
  >│9            str lr, [sp, #0]
   │10
   │11       # Printing The Message
   │12           ldr r0, =HelloWorld
   │13           bl  printf
```

Direct addressing mode

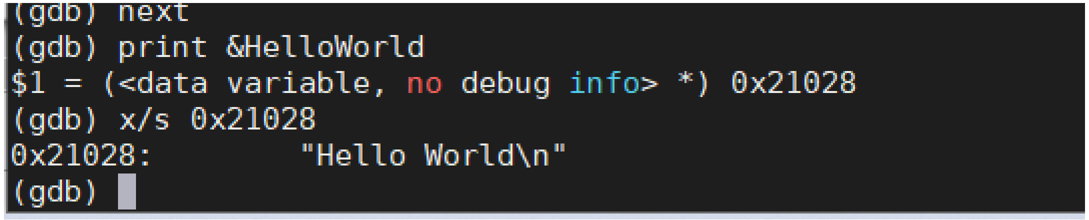
- The following line shows direct addressing of the HelloWorld variable.

```
HelloWorld: .asciz "Hello World\n"
```

- To show this, the print command is run in the gdbtui window, and the print & command prints the address of the HelloWord variable.

- The x/s (eXamine memory, show it as a string) command shows the string is at that address.

Direct address mode example



```
(gdb) next
(gdb) print &HelloWorld
$1 = (<data variable, no debug info> *) 0x21028
(gdb) x/s 0x21028
0x21028:        "Hello World\n"
(gdb) 
```

The label HelloWord is address 0x21028. At address 0x21028 is the string.

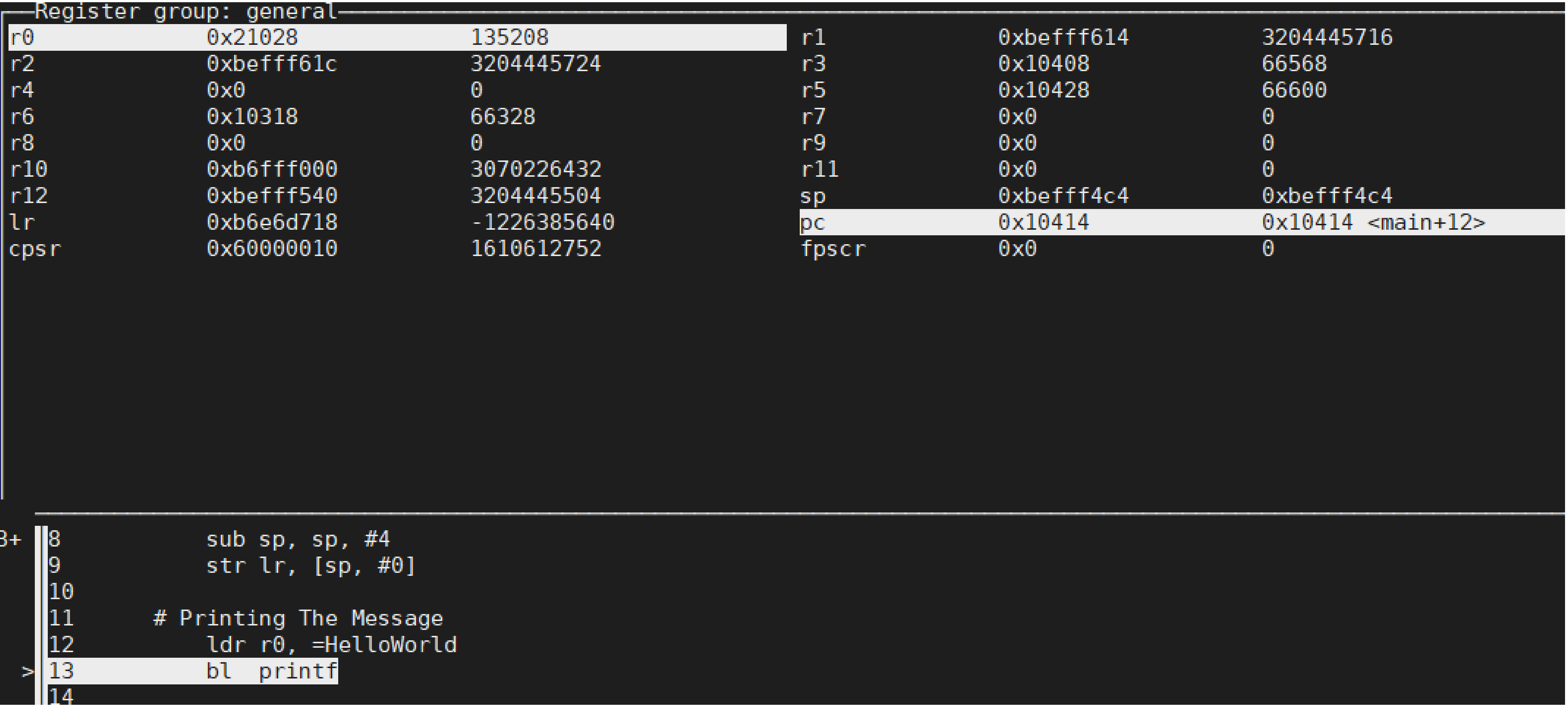
Register indirect mode

- The following line of code illustrates register indirect mode.

```
ldr r0, =HelloWorld
bl  printf
```

- To use the string stored at the address that the HelloWorld label references, the address must be stored in r0.
- The “=HelloWorld” operand retrieves the address for the label “HelloWorld”.
- The ldr stores this address in r0.
- This is shown on the following slide (remember the label has an address of 0x21028.
- r0 now contains an address of a variable, or register indirect address.

Register 0 after running line 12



```
Register group: general
r0             0x21028             135208                  r1             0xbefff614          3204445716
r2             0xbefff61c          3204445724              r3             0x10408             66568
r4             0x0                 0                       r5             0x10428             66600
r6             0x10318             66328                   r7             0x0                 0
r8             0x0                 0                       r9             0x0                 0
r10            0xb6fff000          3070226432              r11            0x0                 0
r12            0xbefff540          3204445504              sp             0xbefff4c4          0xbefff4c4
lr             0xb6e6d718          -1226385640             pc             0x10414             0x10414 <main+12>
cpsr           0x60000010          1610612752              fpscr          0x0                 0

B+ |8           sub sp, sp, #4
   |9           str lr, [sp, #0]
   |10
   |11      # Printing The Message
   |12          ldr r0, =HelloWorld
  >|13          bl  printf
   |14
```

Applying this to scanf and printf – IOExample_3.s

Note: Scanf takes the address of the parameter, but printf takes the value of the parameter.

```
#Scanf
   ldr r0, =input1
   sub sp, sp, #4
         # The following shows register indirect addressing
   mov r1, sp         #←-- pass the address of the stack to scanf, r1 has address
   bl  scanf          #←- scanf will fill the value in at this address
           # The following shows sp is a direct address for variable
          # After execution, r2 has a register direct value
   ldr r2, [sp, #0]   #←- retrieve the value from address sp+0
   add sp, sp, #4

# Printing The Message
   ldr r0, =format1
           # The following shows register direct addressing
   mov r1, r2        #←- Load the value into r1
   bl  printf

.data
num1: .word 0
format1: .asciz "Your Number Is %d \n"
prompt1: .asciz "Enter A Number\n"
input1: .asciz "%d"
```