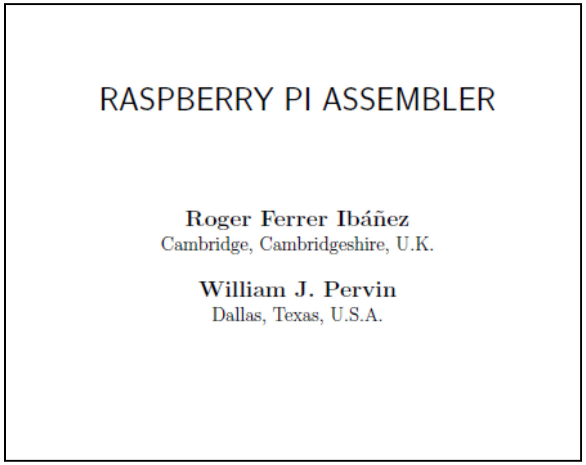
Raspberry Pi Assembler
Assembly and control structures



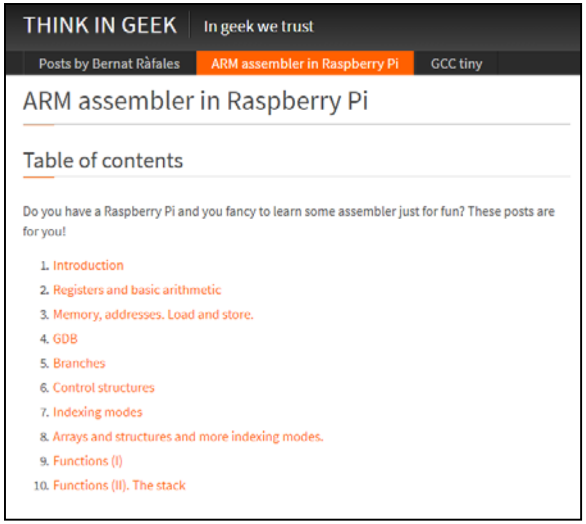
RASPBERRY PI ASSEMBLER

Roger Ferrer Ibáñez
Cambridge, Cambridgeshire, U.K.

William J. Pervin
Dallas, Texas, U.S.A.

Chapter 6: Raspberry Pi Assembler
“Raspberry Pi Assembler” by R. Ferrer and W. Pervin

https://thinkingeek.com/2013/01/20/arm-assembler-raspberry-pi-chapter-6/



THINK IN GEEK In geek we trust

Posts by Bernat Ràfales | ARM assembler in Raspberry Pi | GCC tiny

ARM assembler in Raspberry Pi

Table of contents

Do you have a Raspberry Pi and you fancy to learn some assembler just for fun? These posts are for you!

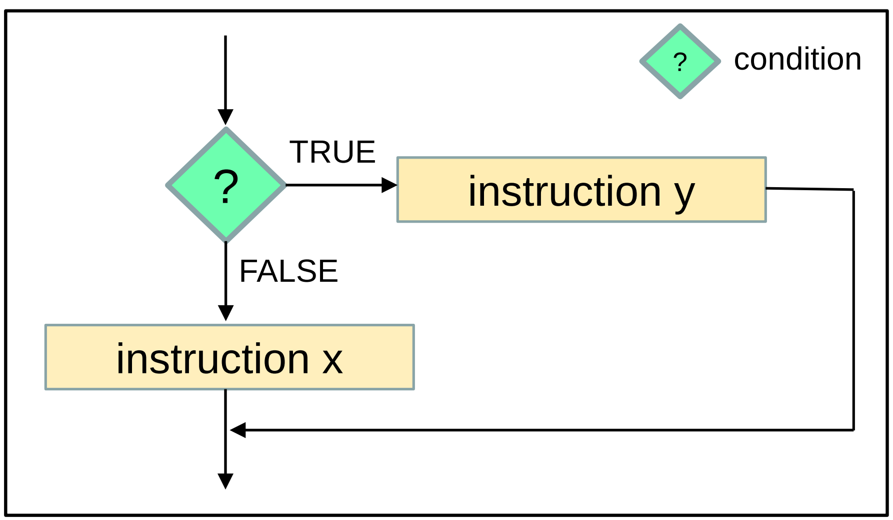
1. Introduction
2. Registers and basic arithmetic
3. Memory, addresses. Load and store.
4. GDB
5. Branches
6. Control structures
7. Indexing modes
8. Arrays and structures and more indexing modes.
9. Functions (I)
10. Functions (II). The stack

Embedded Systems II – EEE3096S
Y. Abdul Gaffar
S. Winberg

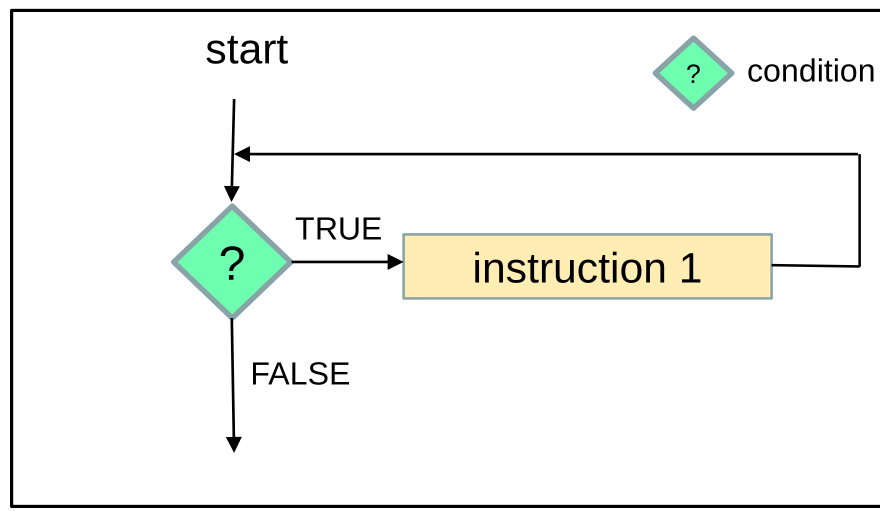
Raspberry Pi Assembler
Assembly and control structures

- Structured programming constructs are generally used to develop a program. These consists of:
  - If, then, else
  - Loops: while loop and for loops
  - …. many more

- Let's look at developing these constructs in assembly language



Flow chart of a 'if-then-else' construct



Flow chart of a 'while' loop

Raspberry Pi Assembler

Control structures: If, then, else

- Generic: If, then, else structure, where E is a condition and S1 and S2 are instructions

```
if (!E) then
   S1
else
   S2
```

- ARM assembler code to implement the If, then, else construct

Choose the appropriate branch suffix so the program will branch to the “else_part” when the condition is TRUE

```
if_eval:
    /* Assembler that evaluates E and updates the cpsr accordingly */
bXX else_part   /* Here XX is the appropriate condition */
then_part:
    /* assembler code for S1, the "then" part */
    b end_of_if
else_part:
    /* assembler code for S2, the "else" part */
end_of_if:
```

Raspberry Pi Assembler
Control structures: If, then, else

- Example:

```
if (r1 == r2)  then
     r0 = 1
else
     r0 = 2
end
```

- ARM assembler code

```
If_eval:      cmp  r1, r2       /* computes  (r1 – r2) and updates the CPSR */
BNE  else                  /* branch taken when condition (Z == 0) is TRUE */

then_part:    mov  r0, #1
              b  end_of_if

else:         mov r0, #2

end_of_if:
```

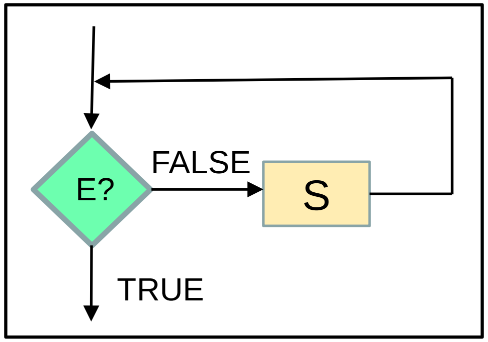
Raspberry Pi Assembler

Control structures: While loop

- Generic while loop structure, where E is a condition and S is an instruction

```
while (!E)
  S
```

while condition E is false,
perform instruction S



- ARM assembler code to implement the while loop construct

```
while_condition:
  /* assembler code to evaluate E and update cpsr */
  bXX end_of_loop   /* If E is true, leave the loop right now */
  /* assembler code for the statement S */
  b while_condition /* Unconditional branch to the beginning */
end_of_loop:
```

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

Raspberry Pi Assembler
Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0     /* sum variable */
r2 = 1     /* counter */

while ( !(r2 > 22) )
    r0 = r0 + r2
    r2 = r2 + 1
end
```

- Assembly code

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0         @ r1 <- 0
    mov r2, #1         @ r2 <- 1
loop:
    cmp r2, #22        @ compare r2 and 22
    bgt end            @ branch if r2 > 22 to end
    add r1, r1, r2     @ r1 <- r1 + r2
    add r2, r2, #1     @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1         @ r0 <- r1
    bx  lr
```

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
    r0 = r0 + r2
    r2 = r2 + 1
end
```

- sum variable r1 is initialised to 0
- counter variable r2 initialised to 1

- Assembly code

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0     /* sum variable */
r2 = 1     /* counter */

while ( !(r2 > 22) )
    r0 = r0 + r2
    r2 = r2 + 1
end
```

- Assembly code

- compute (r2 – 22)
- then, update the CPSR

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Iteration	r1	r2
1	0	1

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
     r0 = r0 + r2
     r2 = r2 + 1
end
```

- branch when r2 > 22
- since r2 = 1, the branch will not take place

- Assembly code

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0          @ r1 <- 0
    mov r2, #1          @ r2 <- 1
loop:
    cmp r2, #22         @ compare r2 and 22
    bgt end             @ branch if r2 > 22 to end
    add r1, r1, r2      @ r1 <- r1 + r2
    add r2, r2, #1      @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1          @ r0 <- r1
    bx  lr
```

Last instruction executed

Iteration	r1	r2
1	0	1

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Iteration	r1	r2
1	1	1

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Last instruction executed

Iteration	r1	r2
1	1	2

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

branch to label ‘loop’

```
/* -- loop01.s */
.text
.global main
main:
     mov r1, #0       @ r1 <- 0
     mov r2, #1       @ r2 <- 1
loop:
     cmp r2, #22      @ compare r2 and 22
     bgt end          @ branch if r2 > 22 to end
     add r1, r1, r2   @ r1 <- r1 + r2
     add r2, r2, #1   @ r2 <- r2 + 1
     b   loop
end:
     mov r0, r1       @ r0 <- r1
     bx  lr
```

Last instruction executed

Iteration	r1	r2
1	1	2

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0     /* sum variable */
r2 = 1     /* counter */

while ( !(r2 > 22) )
    r0 = r0 + r2
    r2 = r2 + 1
end
```

- compute (r2 – 22)
- then, update the CPSR

- Assembly code

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0          @ r1 <- 0
    mov r2, #1          @ r2 <- 1
loop:
    cmp r2, #22         @ compare r2 and 22
    bgt end             @ branch if r2 > 22 to end
    add r1, r1, r2      @ r1 <- r1 + r2
    add r2, r2, #1      @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1          @ r0 <- r1
    bx  lr
```

Iteration	r1	r2
1	1	2
2	1	2

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
    - Use CPU register r1 as the sum variable
    - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

branch to label ‘loop’

```
/* -- loop01.s */
.text
.global main
main:
     mov r1, #0         @ r1 <- 0
     mov r2, #1         @ r2 <- 1
loop:
     cmp r2, #22        @ compare r2 and 22
     bgt end            @ branch if r2 > 22 to end
     add r1, r1, r2     @ r1 <- r1 + r2
     add r2, r2, #1     @ r2 <- r2 + 1
     b   loop
end:
     mov r0, r1         @ r0 <- r1
     bx  lr
```

Last instruction executed

Iteration	r1	r2
1	1	2
2	3	3

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

- compute (r2 – 22)
- then, update the CPSR

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Iteration	r1	r2
1	1	2
2	3	3
3	3	3

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

branch to label ‘loop’

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
     mov r1, #0         @ r1 <- 0
     mov r2, #1         @ r2 <- 1
loop:
     cmp r2, #22        @ compare r2 and 22
     bgt end            @ branch if r2 > 22 to end
     add r1, r1, r2     @ r1 <- r1 + r2
     add r2, r2, #1     @ r2 <- r2 + 1
     b   loop
end:
     mov r0, r1         @ r0 <- r1
     bx  lr
```

Iteration	r1	r2
1	1	2
2	3	3
3	6	4

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Iteration	r1	r2
1	1	2
2	3	3
3	6	4
…	…	…

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

branch to label 'loop'

```
/* -- loop01.s */
.text
.global main
main:
     mov r1, #0         @ r1 <- 0
     mov r2, #1         @ r2 <- 1
loop:
     cmp r2, #22        @ compare r2 and 22
     bgt end            @ branch if r2 > 22 to end
     add r1, r1, r2     @ r1 <- r1 + r2
     add r2, r2, #1     @ r2 <- r2 + 1
     b   loop
end:
     mov r0, r1         @ r0 <- r1
     bx  lr
```

Last instruction executed

Iteration	r1	r2
1	1	2
2	3	3
3	6	4
…	…	…
23	253	23

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- compute (r2 – 22)
- then, update the CPSR

- Assembly code

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
        mov r1, #0          @ r1 <- 0
        mov r2, #1          @ r2 <- 1
loop:
        cmp r2, #22         @ compare r2 and 22
        bgt end             @ branch if r2 > 22 to end
        add r1, r1, r2      @ r1 <- r1 + r2
        add r2, r2, #1      @ r2 <- r2 + 1
        b   loop
end:
        mov r0, r1          @ r0 <- r1
        bx  lr
```

Iteration	r1	r2
1	1	2
2	3	3
3	6	4
…	…	…
23	253	23
24	253	23

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- branch when r2 > 22
- since r2 = 23, the branch will take place

- Assembly code

Last instruction executed

```
/* -- loop01.s */
.text
.global main
main:
      mov r1, #0          @ r1 <- 0
      mov r2, #1          @ r2 <- 1
loop:
      cmp r2, #22         @ compare r2 and 22
      bgt end             @ branch if r2 > 22 to end
      add r1, r1, r2      @ r1 <- r1 + r2
      add r2, r2, #1      @ r2 <- r2 + 1
      b   loop
end:
      mov r0, r1          @ r0 <- r1
      bx  lr
```

Iteration	r1	r2
1	1	2
2	3	3
3	6	4
…	…	…
23	253	23
24	253	23

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0      /* sum variable */
r2 = 1      /* counter */

while ( !(r2 > 22) )
      r0 = r0 + r2
      r2 = r2 + 1
end
```

- Assembly code

r0 = 253

```
/* -- loop01.s */
.text
.global main
main:
     mov r1, #0         @ r1 <- 0
     mov r2, #1         @ r2 <- 1
loop:
     cmp r2, #22        @ compare r2 and 22
     bgt end            @ branch if r2 > 22 to end
     add r1, r1, r2     @ r1 <- r1 + r2
     add r2, r2, #1     @ r2 <- r2 + 1
     b   loop
end:
     mov r0, r1         @ r0 <- r1
     bx  lr
```

Last instruction executed

Iteration	r1	r2
1	1	2
2	3	3
3	6	4
…	…	…
23	253	23
24	253	23

Raspberry Pi Assembler

Example: program using control structures

- Task: write an assembly program to sum all the numbers from 1 to 22
- Approach taken:
  - Use CPU register r1 as the sum variable
  - Use CPU register r2 as a counter

```
r1 = 0     /* sum variable */
r2 = 1     /* counter */

while ( !(r2 > 22) )
    r0 = r0 + r2
    r2 = r2 + 1
end
```

```
$ ./loop01; echo $?
253
```

r0 = 253

program terminates

- Assembly code

```
/* -- loop01.s */
.text
.global main
main:
    mov r1, #0        @ r1 <- 0
    mov r2, #1        @ r2 <- 1
loop:
    cmp r2, #22       @ compare r2 and 22
    bgt end           @ branch if r2 > 22 to end
    add r1, r1, r2    @ r1 <- r1 + r2
    add r2, r2, #1    @ r2 <- r2 + 1
    b   loop
end:
    mov r0, r1        @ r0 <- r1
    bx  lr
```

Last instruction executed

Iteration	r1	r2
1	1	2
2	3	3
3	6	4
…	…	…
23	253	23
24	253	23

Coding up the Collatz conjecture in assembly

Raspberry Pi Assembler
Collatz conjecture

- What is the Collatz conjecture?
  - Given a number *n*,
    - If the number is even, we divide it by 2
    - If the number is odd, we multiply it by 3 and add one

    Keep applying this rule until the final value is 1 and display the number of iterations that was performed.

- Pseudo-code for the Collatz conjecture
  1. Intialise n
  2. While (n != 1), go to step 3, else exit the program
  3. If n is an even number, n = n/2
  4. If n is odd, n = 3*n + 1
  5. Go to step 2

```
n = 123
i = 0

while (n != 1)
          if (n mod 2 == 0)
                n = n/2
           else
                 n = 3*n +1
            end
            i = i + 1
end
```

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123             @ r1 <- 123 a trial number
    mov r2, #0               @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1               @ compare r1 and 1
    beq end                  @ branch to end if r1 == 1

    and r3, r1, #1           @ r3 <- r1 & 1   [mask]
    cmp r3, #0               @ compare r3 and 0
    bne odd                  @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1       @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1   @ r1 <- r1 + (r1 << 1)    [3n]
    add r1, r1, #1           @ r1 <- r1 + 1    [3n+1]

end_loop:
    add r2, r2, #1           @ r2 <- r2 + 1
    b   loop                 @ branch to loop

end:
    mov r0, r2               @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
        if (r1 mod 2 == 0)
            r1 = r1/2
        else
            r1 = 3*r1 +1
        end
        r2 = r2 + 1
end
```

Initialise
- value to evaluate: r1 = 123
- number of iterations: r2 = 0

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123          @ r1 <- 123 a trial number
    mov r2, #0            @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1            @ compare r1 and 1
    beq end               @ branch to end if r1 == 1

    and r3, r1, #1        @ r3 <- r1 & 1  [mask]
    cmp r3, #0            @ compare r3 and 0
    bne odd               @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1    @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1        @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1        @ r2 <- r2 + 1
    b   loop              @ branch to loop

end:
    mov r0, r2            @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
      if (r1 mod 2 == 0)
          r1 = r1/2
      else
          r1 = 3*r1 +1
      end
      r2 = r2 + 1
end
```

- compute r1 - 1
- then, update the CPSR

- Since the result of the last operation is 122 and not 0, the branch does not take place

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1   [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1  @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
    if (r1 mod 2 == 0)
        r1 = r1/2
    else
        r1 = 3*r1 +1
    end
    r2 = r2 + 1
end
```

Check if the value of r1 is an odd number by performing the following operations

- A bitwise AND of r1 with 0x00000001 and put the result into r3.
- If bit0 of r3 is 1, then r1 is odd
- If bit0 of r3 is 0, then r3 is even

Note: bne means if r3 $\neq$ 0, then perform the branch to label odd

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1  [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
        if (r1 mod 2 == 0)
            r1 = r1/2
        else
            r1 = 3*r1 +1
        end
        r2 = r2 + 1
end
```

The operation 3*r1 + 1 is performed using the following steps

- r1 = r1 + 2*r1 to give 3*r1
- then, add one: r1 = r1 + 1

add r1, r1, r1, LSL #1

Logical shift left: fast multiplication 2*r1

r1 = 3*123 + 1
= 370

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1  [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
        if (r1 mod 2 == 0)
              r1 = r1/2
         else
               r1 = 3*r1 +1
         end
         r2 = r2 + 1
end
```

Increment r2 by one and branch to label loop

r1 = 370
r2 = 1

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1  [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
        if (r1 mod 2 == 0)
            r1 = r1/2
        else
            r1 = 3*r1 +1
        end
        r2 = r2 + 1
end
```

- compute r1 - 1
- then, update the CPSR

- Since the result of the last operation is 369 and not 0, the branch does not take place

r1 = 370
r2 = 1

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1   [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)    [3n]
    add r1, r1, #1          @ r1 <- r1 + 1    [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
      if (r1 mod 2 == 0)
            r1 = r1/2
      else
            r1 = 3*r1 +1
      end
      r2 = r2 + 1
end
```

Check if the value of r1 is an odd number by performing the following operations

- A bitwise AND of r1 with 0x00000001 and put the result into r3.
- If bit0 of r3 is 1, then r1 is odd
- If bit0 of r3 is 0, then r3 is even

Note: bne means if r3 ≠ 0, then perform the branch to label odd

r1 = 370
r2 = 1

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1  [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
    if (r1 mod 2 == 0)
        r1 = r1/2
    else
        r1 = 3*r1 +1
    end
    r2 = r2 + 1
end
```

mov r1, r1, r1, ASR #1

Arithmetic shift right: fast division r1/2

r1 = 185
r2 = 1

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123           @ r1 <- 123 a trial number
    mov r2, #0             @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1             @ compare r1 and 1
    beq end                @ branch to end if r1 == 1

    and r3, r1, #1         @ r3 <- r1 & 1  [mask]
    cmp r3, #0             @ compare r3 and 0
    bne odd                @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1     @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1         @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1         @ r2 <- r2 + 1
    b   loop               @ branch to loop

end:
    mov r0, r2             @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
      if (r1 mod 2 == 0)
          r1 = r1/2
      else
          r1 = 3*r1 +1
      end
      r2 = r2 + 1
end
```

Increment r2 by one and branch to label loop

r1 = 185
r2 = 2

Raspberry Pi Assembler

Collatz conjecture: assembly program

```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1  [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
r1 = 123
r2 = 0

while (r1 != 1)
        if (r1 mod 2 == 0)
             r1 = r1/2
         else
              r1 = 3*r1 +1
         end
         r2 = r2 + 1
end
```

r2	r1
0	123
1	370
2	185
…	…
46	1

Raspberry Pi Assembler

Collatz conjecture: assembly program

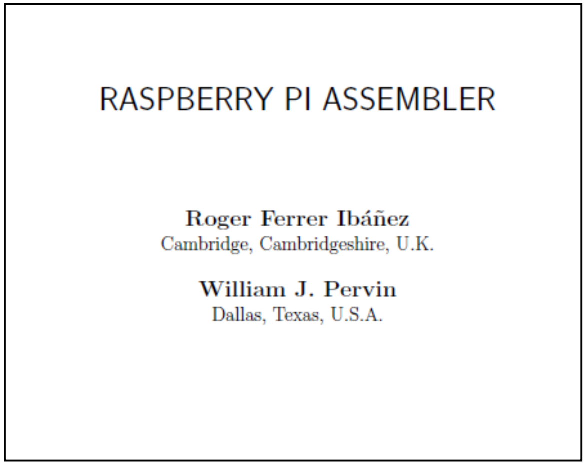
```
/* -- collatz.s */
.text
.global main
main:
    mov r1, #123            @ r1 <- 123 a trial number
    mov r2, #0              @ r2 <- 0  the #  of steps
loop:
    cmp r1, #1              @ compare r1 and 1
    beq end                 @ branch to end if r1 == 1

    and r3, r1, #1          @ r3 <- r1 & 1  [mask]
    cmp r3, #0              @ compare r3 and 0
    bne odd                 @ branch to odd if r3 != 0
even:
    mov r1, r1, ASR #1      @ r1 <- (r1 >> 1)  [divided by 2]
    b   end_loop
odd:
    add r1, r1, r1, LSL #1 @ r1 <- r1 + (r1 << 1)   [3n]
    add r1, r1, #1          @ r1 <- r1 + 1   [3n+1]

end_loop:
    add r2, r2, #1          @ r2 <- r2 + 1
    b   loop                @ branch to loop

end:
    mov r0, r2              @ number of steps
    bx  lr
```

```
$ ./collatz; echo $?
46
```

Total number of iterations is 46

r2	r1
0	123
1	370
2	185
…	…
46	1

Raspberry Pi Assembler
ARM Assembly syntax

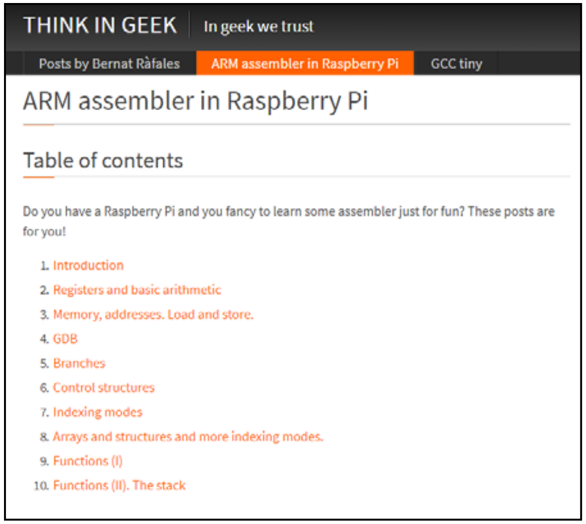

RASPBERRY PI ASSEMBLER

Roger Ferrer Ibáñez
Cambridge, Cambridgeshire, U.K.

William J. Pervin
Dallas, Texas, U.S.A.

Chapter 7: Raspberry Pi Assembler
“Raspberry Pi Assembler” by R. Ferrer and W. Pervin

https://thinkingeek.com/2013/01/26/arm-assembler-raspberry-pi-chapter-7/


THINK IN GEEK In geek we trust

Posts by Bernat Ràfales | ARM assembler in Raspberry Pi | GCC tiny

ARM assembler in Raspberry Pi

Table of contents

Do you have a Raspberry Pi and you fancy to learn some assembler just for fun? These posts are for you!

1. Introduction
2. Registers and basic arithmetic
3. Memory, addresses. Load and store.
4. GDB
5. Branches
6. Control structures
7. Indexing modes
8. Arrays and structures and more indexing modes.
9. Functions (I)
10. Functions (II). The stack

Raspberry Pi Assembler
ARM assembly syntax

- Let's review the syntax of ARM assembly instructions, so we can understand more advanced assembly instructions
- Most ARM instructions use one of the following three formats:

1) Instruction Rd, source 2 [example: MOV r0, #2]

$\underbrace{\text{Instruction}}_{\text{operation}}\ \underbrace{\text{Rd,}}_{\text{destination CPU register}}\ \underbrace{\text{source 2}}_{\text{source CPU register}}$

2) Instruction Rd, Rn, source2 [example: ADD r2, r1, r0]

$\underbrace{\text{Instruction}}_{\text{operation}}\ \underbrace{\text{Rd,}}_{\text{destination CPU register}}\ \underbrace{\text{Rn,}}_{\text{source CPU register}}\ \underbrace{\text{source2}}_{\text{CPU register or immediate value}}$

Raspberry Pi Assembler

ARM assembly syntax

- Let's review the syntax of ARM assembly instructions, so we can understand more advanced assembly instructions
- Most ARM instructions use one of the following three formats:

1) $\underbrace{\text{Instruction}}_{\text{operation}}\ \underbrace{\text{Rd,}}_{\text{destination CPU register}}\ \underbrace{\text{source 2}}_{\text{source CPU register}}$ [example: MOV r0, #2]

2) $\underbrace{\text{Instruction}}_{\text{operation}}\ \underbrace{\text{Rd,}}_{\text{destination CPU register}}\ \underbrace{\text{Rn,}}_{\text{source CPU register}}\ \underbrace{\text{source2}}_{\text{CPU register or immediate value}}$ [example: ADD r2, r1, r0]

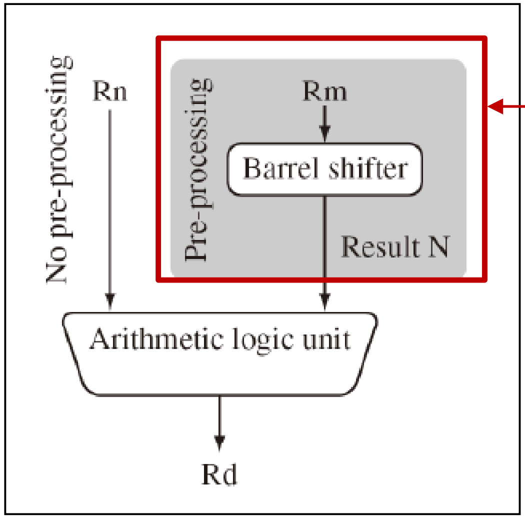
3) $\underbrace{\text{Instruction}}_{\text{operation}}\ \underbrace{\text{Rd,}}_{\text{destination CPU register}}\ \underbrace{\text{Rn,}}_{\text{source CPU register}}\ \underbrace{\text{Rm,}}_{\text{CPU register}}\ \underbrace{\text{operation on Rm}}_{\text{shifting operation performed on Rm}}$

[example: add r1, r2, r3, LSL #1]

Raspberry Pi Assembler
ARM assembly syntax

- Let's look at format 3) in more detail

Instruction	Rd,	Rn,	Rm,	operation on Rm
operation	destination CPU register	source CPU register	CPU register	Shifting operation performed on Rsource2



the ARM instruction set can perform a shifting operation on Rm and then pass this result to the ALU

Raspberry Pi Assembler
ARM assembly syntax

- Let’s look at format 3) in more detail by looking at an example

add r1, r2, r3, LSL #1

means: the logical shift left by one needs
to be applied to r3 before the add operation

Raspberry Pi Assembler

ARM assembly syntax

- Let's look at format 3) in more detail by looking at an example

add r1, r2, $\underbrace{\text{r3, LSL \#1}}$

means: the logical shift left by one needs to be applied to r3 before the add operation

[r1 = r2 + $\underbrace{\text{(r3 << 1)}}$]

- First perform a logical shift left of r3
- Then, add r2 with the shifted value of r3 and put the result into r1

Raspberry Pi Assembler
ARM assembly syntax

- Let’s look at format 3) in more detail by looking at an example

add r1, r2, r3, LSL #1

means: the logical shift left by one needs
to be applied to r3 before the add operation

[r1 = r2 + (r3 << 1)]

- First perform a logical shift left of r3
- Then, add r2 with the shifted value of r3 and put the result into r1

- Note: the ARM instruction set does not have a shift right or a shift left instruction by itself. In order to shift the contents of a register, we can use the MOV instruction. Example below: logical shift left the contents of r3:

mov r3, r3, LSL #1

[r3 = (r3 << 1)]

Raspberry Pi Assembler
ARM assembly syntax: shift operations

Logical Shift

(unsigned numbers)

- **n-bit left**: multiply by 2^n
- **n-bit right**: divide by 2^n

LSL #n Logical Shift Left. Shifts bits n times left. The n leftmost bits are lost and the n rightmost are set to zero.

LSL Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

LSR #n Logical Shift Right. Shifts bits n times right. The n rightmost bits are lost and the n leftmost bits are set to zero,

LSR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

ASR #n Arithmetic Shift Right. Like LSR but the leftmost bit before shifting is used instead of zero in the n leftmost ones.

ASR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

ROR #n ROtate Right. Like LSR but the n rightmost bits are not lost but pushed onto the n leftmost bits

ROR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

Raspberry Pi Assembler

ARM assembly syntax: shift operations

Arithmetic Shift

(signed numbers)

- **n-bit left**: multiply by 2^n
- **n-bit right**: divide by 2^n

LSL #n Logical Shift Left. Shifts bits n times left. The n leftmost bits are lost and the n rightmost are set to zero.

LSL Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

LSR #n Logical Shift Right. Shifts bits n times right. The n rightmost bits are lost and the n leftmost bits are set to zero,

LSR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

ASR #n Arithmetic Shift Right. Like LSR but the leftmost bit before shifting is used instead of zero in the n leftmost ones.

ASR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

ROR #n ROtate Right. Like LSR but the n rightmost bits are not lost but pushed onto the n leftmost bits

ROR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

Raspberry Pi Assembler

ARM assembly syntax: shift operations

LSL #n Logical Shift Left. Shifts bits n times left. The n leftmost bits are lost and the n rightmost are set to zero.

LSL Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

LSR #n Logical Shift Right. Shifts bits n times right. The n rightmost bits are lost and the n leftmost bits are set to zero,

LSR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

ASR #n Arithmetic Shift Right. Like LSR but the leftmost bit before shifting is used instead of zero in the n leftmost ones.

ASR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

Circular shift right

ROR #n ROtate Right. Like LSR but the n rightmost bits are not lost but pushed onto the n leftmost bits

ROR Rsource3 Like the previous one but instead of an immediate value the lower byte of the register specifies the amount of shifting.

Raspberry Pi Assembler

ARM assembly syntax: examples of shifting

- mov r1, r2, LSL #1 [r1 = 2 x r2]
- mov r1, r2, LSL #2 [r1 = 4 x r2]
- mov r1, r3, ASR #3 [r1 = r3 ÷ 8]
- mov r3, #4
  mov r1, r2, LSL r3 [r1 = 16 x r2]

- Let's observe how we can use this type of operation to compute multiplication by a number that is not a power of 2, ie not 2, 4, 8, 16, 32, …

Raspberry Pi Assembler
Non-power of 2 multiplication

- Addition, subtraction and shifting operations can be used to compute the multiplication of a number that is not a power of 2

- Example: compute r1 = 7 x r2

Raspberry Pi Assembler

Non-power of 2 multiplication

- Addition, subtraction and shifting operations can be used to compute the multiplication of a number that is not a power of 2

- Example: compute r1 = 7 x r2

$$\text{rsb r1, r2, } \underbrace{\text{r2, LSL \#3}}_{8 \times r2}$$

Raspberry Pi Assembler

Non-power of 2 multiplication

- Addition, subtraction and shifting operations can be used to compute the multiplication of a number that is not a power of 2

- Example: compute r1 = 7 x r2

$$\text{rsb r1, } \overbrace{\text{r2, } \underbrace{\text{r2, LSL \#3}}_{8 \times r2}}^{r1 = (8 \times r2) - r2 \; = 7 \times r2}$$

Raspberry Pi Assembler
Non-power of 2 multiplication

- Addition, subtraction and shifting operations can be used to compute the multiplication of a number that is not a power of 2

- Example: compute r1 = 7 x r2

$$\text{rsb}\ \overbrace{\text{r1, r2, }\underbrace{\text{r2, LSL \#3}}_{8 \text{ x } r2}}^{r1 = (8 \text{ x } r2) - r2 \;=\; 7 \text{ x } r2}$$

- Note: using shifting operations to perform multiplication uses fewer computations than the ***mult*** (ie. multiply) assembly instruction