August 30, 2021

0.5
0.5000000001

- Review of number system
  - 10, 2, 16
    - 10 <-> 2 <-> , floating pt , overflow signed/unsigned arithmetic

3.1415 -> [ ] 2binary

- 1 -> 1.5 writing ARM assembly

Number systems

0, ..., 9, A, B, C, D, E, F
0, ..., 9, 10, 11, 12, 13, 14, 15

Base 10	Base 2	Base 16
0...9	0, 1	0...9, A, B, C, D, E, F

32-bit

118	⟷	6 5 4 3 2 1 0 1 1 1 0 1 1 0	⟷	
$_{10}$		$_2$		$_{16}$

$$2\overline{)118} \quad R\boxed{0}$$
$$2\overline{)59} \quad R\boxed{1}$$
$$2\overline{)29} \quad R\boxed{1}$$
$$2\overline{)14} \quad R\boxed{0}$$
$$2\overline{)7} \quad R\boxed{1}$$
$$2\overline{)3} \quad R\boxed{1}$$
$$2\overline{)1} \quad R\boxed{1}$$

$$\left[0*2^{3}+1*2^{4}+1*2^{5}+0*2^{6}+1*2^{7}+1*2^{8}+1*2^{9}\right]\times 2^{3}$$

$$0+2+4+0+16+32+64$$

96

112

116

118

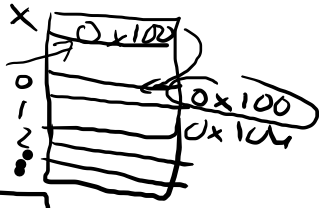
9	8	7	6	5	4	3	2	1	0
1	1	1	0	1	1	0	0	0	0

int n = 118;

n = n << 3;

118 * 8 → ALU

int x[5]; →



$\boxed{118}_{10} \longleftrightarrow \boxed{\cdots 0\ 7\ 6}_{16}$ (hexadecimal)

$\boxed{0x076}$

$$\begin{array}{r} 7 \\ 16\overline{)118} \\ 112 \\ \hline 6 \end{array} \quad R\ \boxed{6}$$

$$6 \times 16^0 + 7 \times 16^1$$

$$6 + 112 = 118_{10}$$

$$\begin{array}{r} 0 \\ 16\overline{)7} \end{array} \quad R\ \boxed{7}$$

$$2 \longleftrightarrow \boxed{4}_{2^3}$$

$$7 \rightarrow 10 \rightarrow 2$$

$$\boxed{\overset{3\ 2\ 1\ 0}{0111}\ \overset{3\ 2\ 1\ 0}{0\ 1\ 1\ 0}}_2 \rightarrow \boxed{7\ 6}_{16}$$

$2^2 + 2^1 + 2^0$

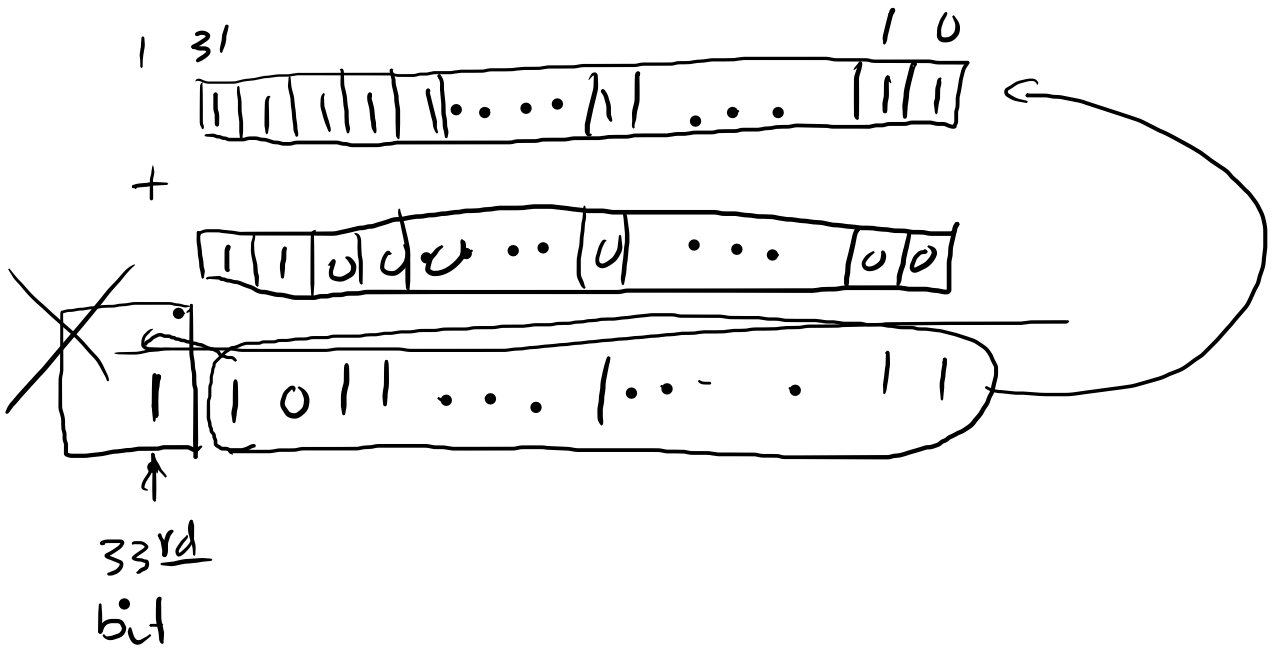
$4 + 2 + 1$

7

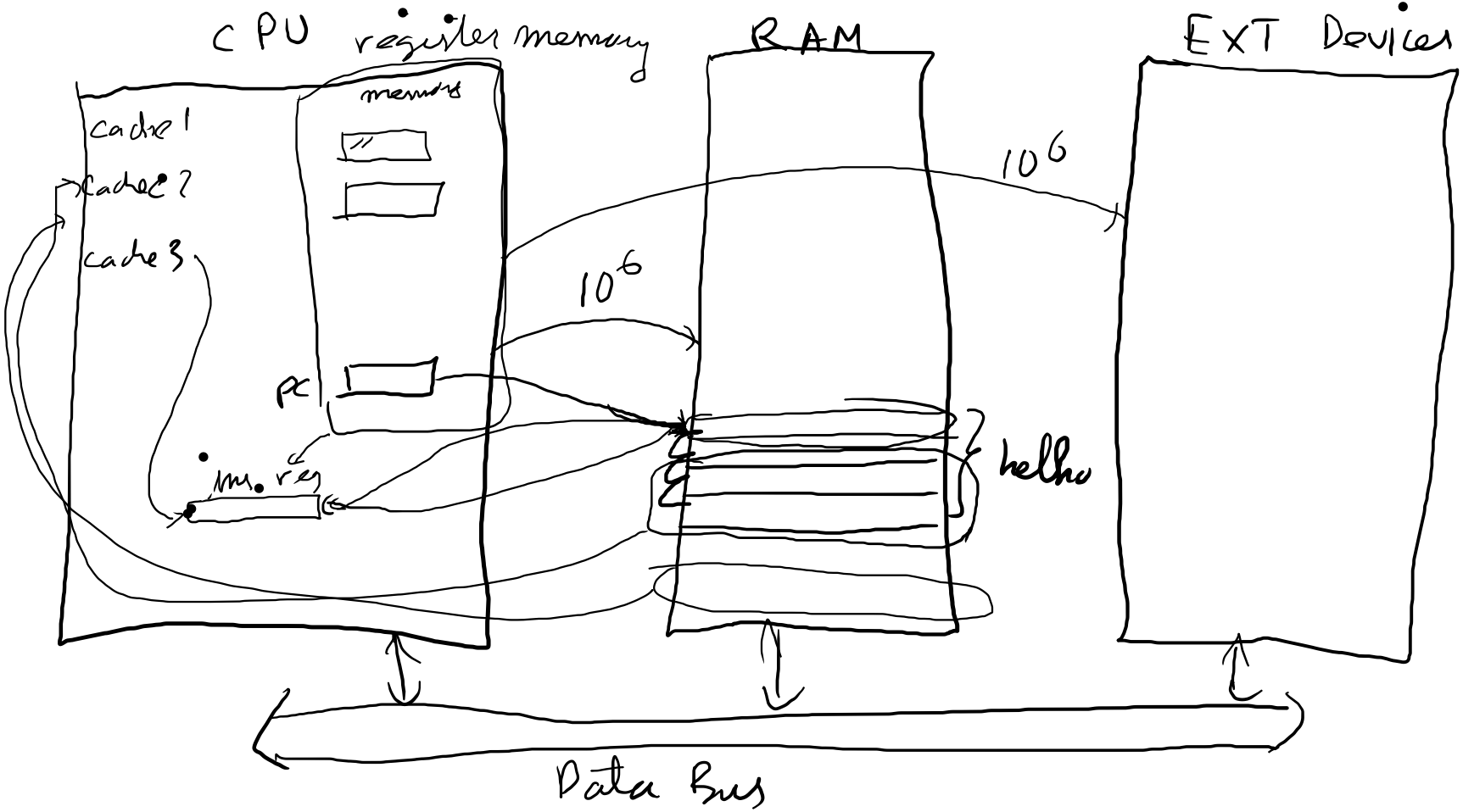
$1 \times 2^1 + 1 \times 2^2$

$2 + 4 = 6$

overflow, unsigned



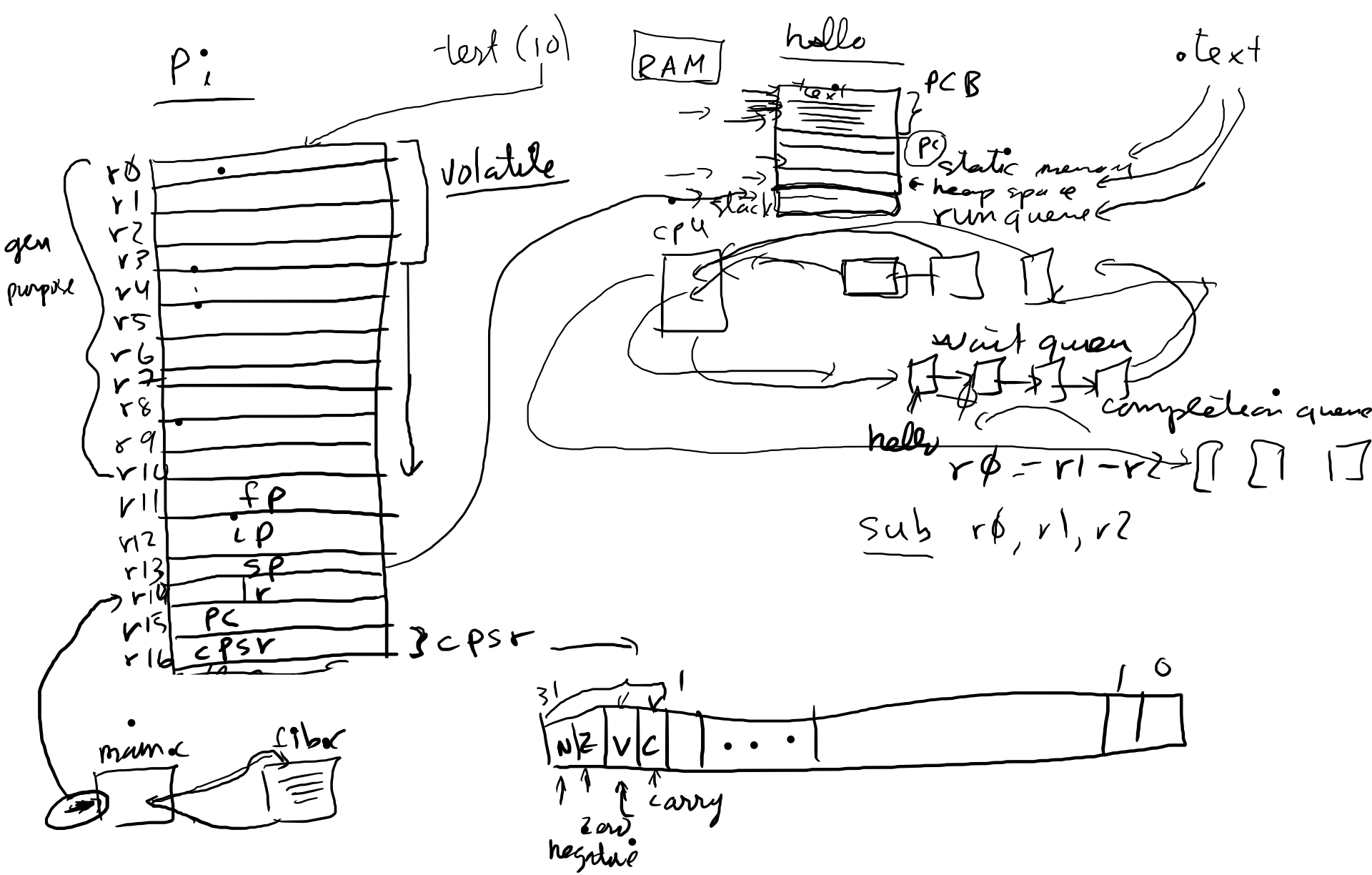
Assembler programming (ARM)


CPU
register memory
RAM
EXT Devices
memory
cache 1
cache 2
cache 3
PC
ins. reg
10^6
10^6
hello
Data Bus

>> hello

010 10 ...

⋮


P1
text (10)
volatile
gen purpose
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13
r14
r15
r16
fp
ip
sp
r
pc
cpsr
cpsr
main
fibo
RAM
hello
text
PCB
pc
static memory
heap space
stack
run queue
cpu
wait queue
completion queue
hello
r0 = r1 - r2
sub r0, r1, r2
31
1
0
N
Z
V
C
carry
zero
negative

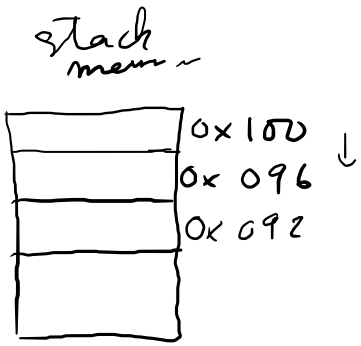
Assembler Coding

(1) architecture of processor

- .cpu cortex-a53 @ Pi 4 → .cpu cortex-a72
- .fpu neon-fp-armv8

(2) .data

@ global var



(3) .text

- .align 2 → memory addressing is on block of 4 bytes
- .global main
- .type main, %function @ differentiate main as a function vs. a simple label

ARM assembler

main:

whileloop: @ address in memory 0x500

alpha numeric

_

$

b whileloop

goto

while (cond)

{

}

Ex: write an ARM assembly prog. to calculate modulo.

5 % 3 → 2

modulo.s

text file

```
@ Define architecture
.cpu cortex-a53
.fpu neon-fp-armv8
@ Define data constants
@ printf("The modulo = %d\n", a)
.data
output: .asciz "The modulo = %d\n"
```

$$\begin{array}{r} 19 \\ 4\overline{)79} \\ 4 \\ \hline 39 \\ 36 \end{array}$$

mov dest, source

- .text
- .align 2
- .global main
- .type main, % function ←

main: @ starting pt. of function (address)

@ save the LR value for every function

@ save LR → r4 (r0 → r10)

mov r4, lr @ r4 = lr

mov r5, #79 @ r5 = 79

• immediate

mov r6, #4 @ r6 = 4

r5 r6

↑ ↑

79 % 4

calculate ~~5 % 3~~

79 % 4

① 79 / 4 = 19

② 19 * 4 = 76 4×19 = 76

③ 79 − 76 = $\boxed{3}$

udiv dest, source1, source2

dest = source1 / source2

@ 79 = r5, r6 = 4

@ 79 / 4 : r5 / r6 → r7

udiv r7, r5, r6 @ r7 = r5 / r6

@ [79/4] * 4

@ r7 * r6 → r7

mul r7, r7, r6 @ r7 = r7 * r6

@ 79 - 76

@ r5 - r7 → r7

Sub r7, r5, r7 @ r7 = r5 - r7

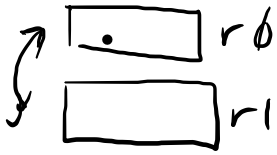
@ r7 = 79 % 4

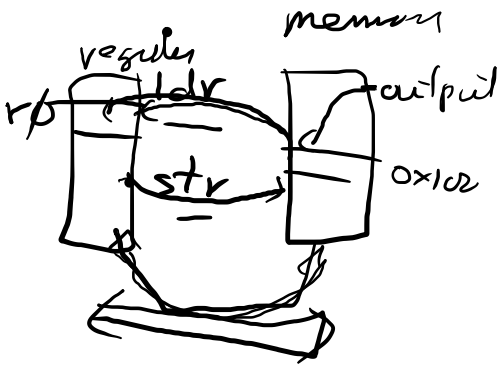
r0 r1

@ printf("The modulo = %d \n", r7)

mov r1, r7 @ r1 = r7

@ r0 = "This modulo = %d \n" ← output



mov (copy between registers)

```
ldr r0, =output
@ printf(r0, r1)
bl printf

@ quit program
@ reset lr
@ lr <-> r4

mov lr, r4   @ lr = r4, restores lr
bx lr   @ go back to where this function was last called
```