



SWINBURNE
UNIVERSITY OF
TECHNOLOGY

COS10004 Computer Systems

Lecture 7.3 ARM Assembly Programming – Some Programming Basics

CRICOS provider 00111D

Dr Chris McCarthy

_start:

ARM ASSEMBLY PROGRAMMING

- Let's write a simple program !

write a program that takes two numbers, adds them together, and stores the answer

ARM ASSEMBLY PROGRAMMING

In a language like C, or Java, this might look something like:

```
int answer = 10 + 4;
```

Pretty simple really ... but let's unpack what is happening

ARM ASSEMBLY PROGRAMMING

```
int answer = 10 + 4;
```

Step 1 – the values 10 and 4 need to be represented somewhere in memory – probably registers

ARM ASSEMBLY PROGRAMMING

```
int answer = 10 + 4;
```

Step 2 – this plus sign needs to be mapped to some sort of instruction that tells the CPU/ALU to use it's Adder (remember the adder circuit!) on these values

ARM ASSEMBLY PROGRAMMING

```
int answer = 10 + 4;
```

Step 3 – the output of this “adder” must be stored somewhere – again, probably a register

ARM ASSEMBLY PROGRAMMING

```
int answer = 10 + 4;
```

Step 4 – define a location in memory big enough to hold an integer, and give it a name

ARM ASSEMBLY PROGRAMMING

```
int answer = 10 + 4;
```

Step 5 – store the result of the addition, currently in a register, in the location associated with the symbolic name “*answer*”

ARM ASSEMBLY PROGRAMMING

```
int answer = 10 + 4;
```

So yeah – simple!



ARM ASSEMBLY PROGRAMMING

So this single line of C code actually entails a bunch of steps.

In assembly programming, we have to define these steps explicitly

In this instance we are going to need 3 different ARM instructions:

- MOV, ADD, and STR

IN ARM ASSEMBLY

```
MOV R0, #10 ; Step 1a:
```

```
MOV R1, #4 ; Step 1b:
```

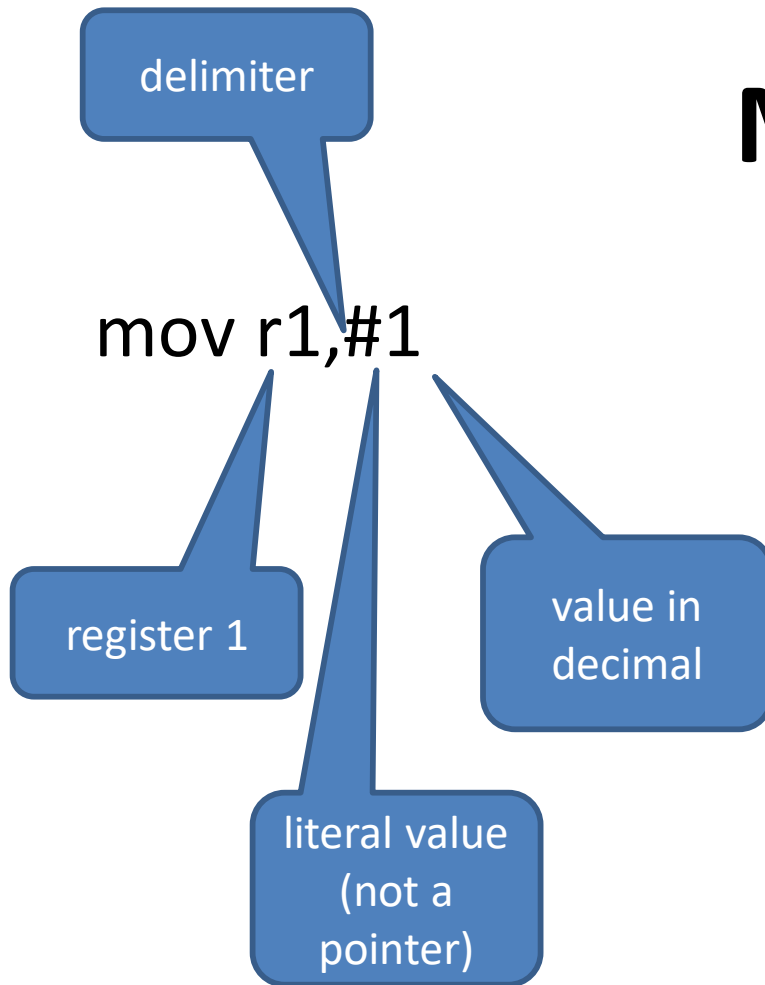
ADD R3, R0, R1 ; Step 2 and Step 3

STR R3, answer ; Step 5

HALT ; STOP!

answer: .word 0 ; Step 4

MOVE



- load register 1 with the value 1*

*not all numbers can be used

ADD v1

- add r2,#1

add 1 to r2

add

to register
2

$r2 = r2 + 1$
 $r2++$

ADD v2

- add r2,r6

add r6 to r2


add

to register
2

$$r2 = r2 + r6$$

ADD v3

- add r2,r5,r6



add r6 and r5
and put into r2

$$r2 = r5 + r6$$

SUB (WORKS THE SAME AS ADD)

- `sub r2,#1`

subtract

to register
2

subtract 1
from r2

'dec' in
some asm

$r2 = r2 - 1$
`r2--`

LABELS

Labels are used to give meaningful names for:

- the location of instructions for branching,
- specify locations of data for loading and storing.

Labels can be made up of any upper or lower case alpha-numeric characters

They must start with a letter

Labels essentially allow programmers to assign meaningful names to memory address locations that will be used often.

This is how we define variables (amongst other things we'll talk about later) in assembly code.

Labels definitions are looked at by the assembler before the executable code

LABELS

Colon indicates
it's a label
definition

someplace: .word 0

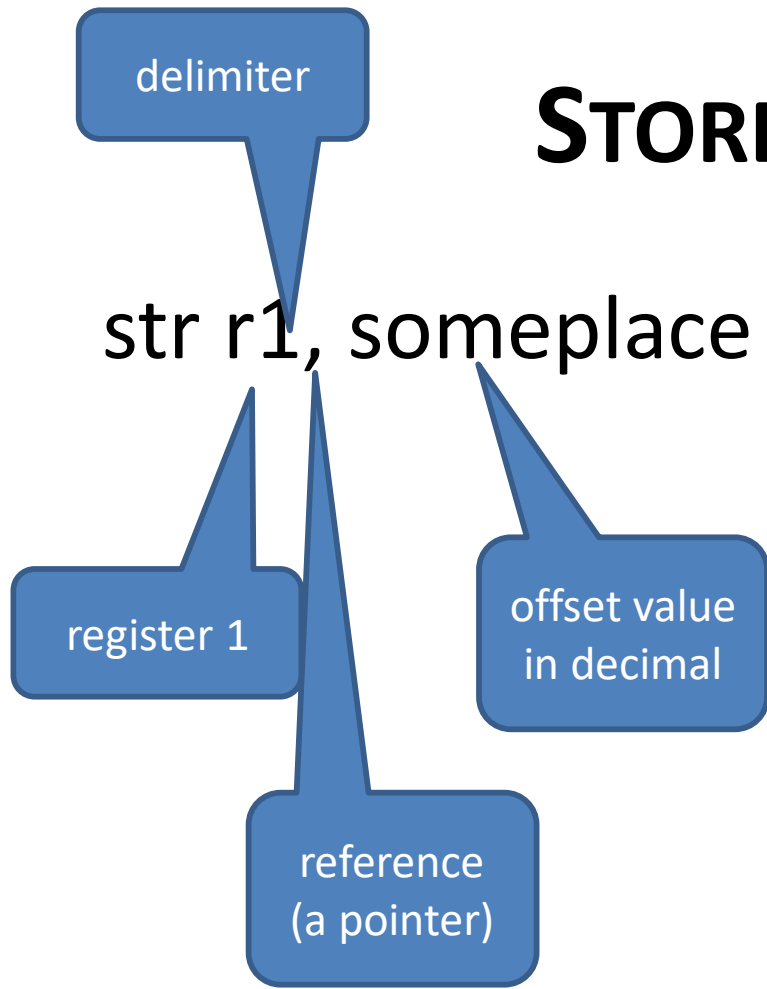
Label name
(whatever
you want)

Type (i.e. size)
of data to be
stored

Value to
initialise
memory
location with

- Define “someplace” as a label for a memory location holding a word value of 0

STORE REGISTER



- write value in `r1` into the memory location associated with the label “someplace”

STORE REGISTER

We'll come back to labels, memory operations and memory addressing later!

STORE REGISTER

We'll come back to labels, memory operations and memory addressing later!

Let's take a look at the code in ARMLite

LETS TAKE A LOOK IN LOGISIM

SUMMARY

- ASM programming represents the lowest level of human readable programming
- Near 1-to-1 match with machine instructions
- RISC versus CISC
- ASM programming is useful for:
 - Hardware specific programming
 - Optimising code for performance
 - Reverse engineering and analysing code behaviour