

Bring the power of Mentimeter to PowerPoint



Seamlessly embed your favorite Menti slide without changing windows.



Edit and do a lot more on Mentimeter.com and sync in real-time.



Log in to use Mentimeter

Work email

Your password

Log in

[Forgot password](#)

[Log in with SSO](#)

or



Don't have an account? [Sign up](#)

ARM Assembly – Arithmetic and Logical Operators

Haris Rotsos

Outline

Introduction to Assembly

- Instructions and operands
- Registers
- Arithmetic operators

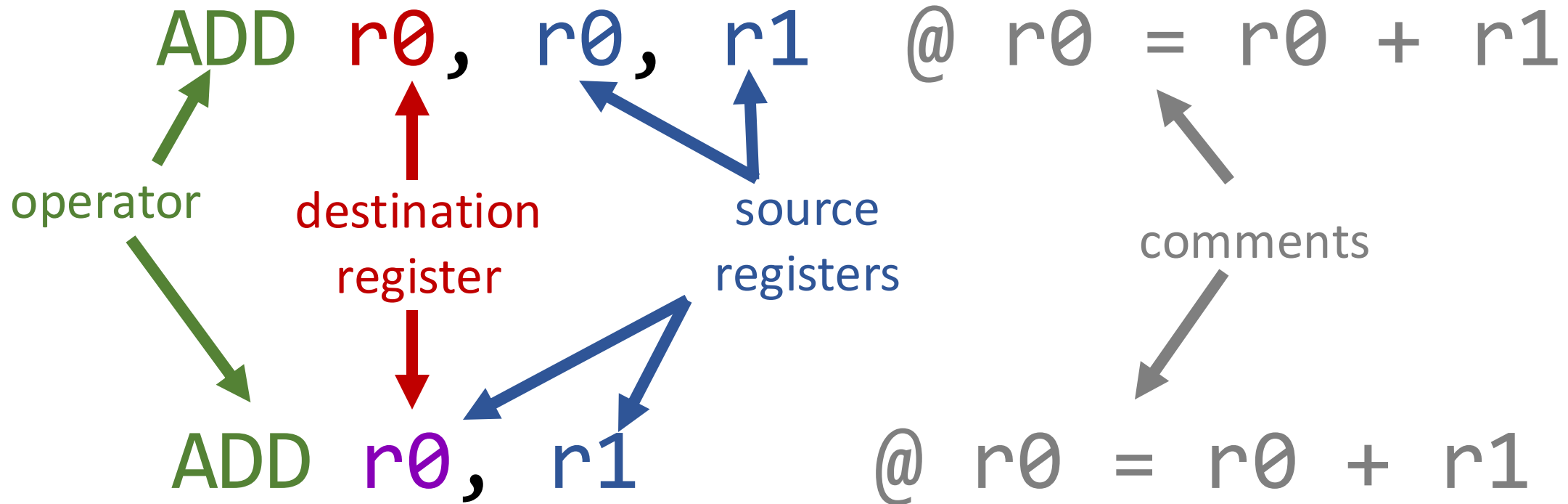
Today we will learn

- How to run our first program.
- Learn our first ARM instructions.

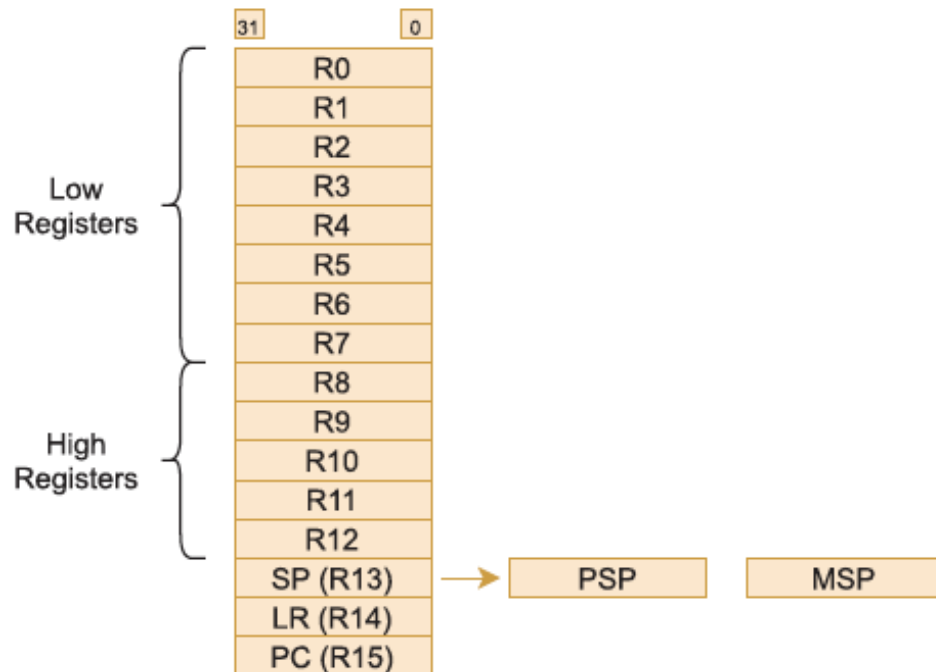
Assembly Instruction

- An instruction is the **most basic unit of computer processing**
 - **Instructions** are words in the language of a computer
 - **Instruction Set Architecture** (ISA) is the vocabulary
- The language of the computer can be written as
 - **Machine language**: Computer-readable representation (that is, 0's and 1's)
 - **Assembly language**: Human-readable representation
- We will study ARM (in detail)
- Principles are similar in all ISAs (x86, SPARC, RISC-V, ...)

Instruction Format




CPU Registers







- Registers are represented with a 4-bit number.
- **R0–R7**: lower registers (bit-4 = 0)
- **R8-R12**: higher registers (bit-4=1)
 - specific instructions store data to them automatically.
- **R13** is the Stack Pointer (SP)
- **R14** is the Link Register (LR)
- **R15** is the program counter (PC)
- **Current Program Status Register (CPSR)**

Example repository


A


Arm Assembly Examples 


 main 


arm-assembly-examples /  


Find file

Edit 







Code 


 Minor fix in comment for lec2
Rotsos, Charalampos authored 5 minutes ago

Verified 

fd705eb9 

History

Name	Last commit	Last update
 lec2	Minor fix in comment for lec2	5 minutes ago
 lec3	Fixing code for lecture 4	1 month ago
 lec4	Fixing code for lecture 4	1 month ago
 .gitignore	Adding gitignore file	1 month ago
 README.md	Fixing code for lecture 4	1 month ago
 microbit-debug.md	Adding top-level readme file	1 month ago

 README.md

Arm Assembly Example

This repository contains the code examples used during the lectures of the Assembly part of the SCC.131 course at Lancaster University. Each repository folder contains the code used during the live coding of each lecture.

In order to load the code in your computer, follow these steps:

1. Clone the repository:

```
git clone https://github.com/yourusername/scc131-examples.git
```

2. Navigate to a lecture repository directory:

Project information

12 Commits

1 Branch

0 Tags

5 KiB Project Storage

README

Auto DevOps enabled

+ Add LICENSE

+ Add CHANGELOG

+ Add CONTRIBUTING

+ Add Kubernetes cluster

+ Add Wiki

+ Configure Integrations

Created on

December 20, 2024

<https://scc-source.lancs.ac.uk/scc.Y1/scc.131/arm-assembly-examples>

Move

- Useful to copy data between registers, or set an immediate to a register.

`mov r1, r2`

“mov” instruction →
destination register →
source registers →

Copy value of r2 in r1

`mov r1, #77`

immediate →

Set r1 to the value 77

Adding

add r1, r1, r2

“add” instruction

destination register

source registers



equivalent C
statement
 $j = j + k;$

register mapping
 $j: r1$
 $k: r2$

Subtracting

subtract instruction

sub r1, r1, r2 or sub r1, r2

destination register

source registers

equivalent C
statement
 $j = j - k;$

register mapping

j: r1

k: r2

Complex Calculations

add r5,r1,r2

add r3,r4

sub r5,r3

equivalent C
statement

$f = (g + h) - (i + j);$

register mapping

f: r5

g: r1

h: r2

i: r3

j: r4

temp: r5, r3

Complex calculations must be broken down in smaller steps → compiler !

Let's test this...

R1 = 10, r2 = 20, r3 = 30

Add r2, r3

Mov r1, r3

Sub r1, r2, r3

Bring the power of
Mentimeter to
PowerPoint



Seamlessly embed your favorite Menti slide without changing windows.



Edit and do a lot more on
Mentimeter.com and sync in real-time.



Log in
to use Mentimeter

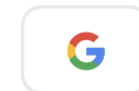
Work email

Your password

Log in

[Forgot password](#) [Log in with SSO](#)

or



Number Encoding

- The "add" instruction assumes that its operands are encoded using **two's complement**
- Since each register holds a 32-bit number:

$-(2^{31})$
-2147483648 to +2147483647

$(2^{31}) - 1$

- Different instructions can have different expectations for how numbers are mapped or coded in registers.

Two's complement

- Positive number is directly converted
- To obtain negative number flip bits and add 1
- To obtain the corresponding positive value flip bits again and add 1

5 → 0000 0101

flip → 1111 1010

add 1 → 1111 1011

-5 → 1111 1011

flip → 0000 0100

add 1 → 0000 0101

5 → 0000 0101

When things go bad

- Sometimes, an arithmetic operation has side-effects.
 - Carry bit.
 - Overflow.
 - Comparison.
- ARM uses a special register (CPSR) to record arithmetic side-effects from instructions.
- We will discuss this a lot when we get to loop and if statements.

Current Program Status Register (CPSR)

- CPSR is an ARM register that records the state of the program.
 - Arithmetic instructions will affect its value every time.
- **N bit - "negative flag"**: instruction result was negative.
- **Z bit - "zero flag"**: instruction result was zero.
- **C bit - "carry flag"**: Instruction causes a carry-out or borrow.
- **V bit - "overflow flag"**: Instruction produces an overflow in 2's complement numbers.



Integer arithmetic operations in ARM

Instruction	Rd	Rn	Rm	imm	Restrictions
ADCS	R0-R7	R0-R7	R0-R7	-	$Rd = Rn + Rm + \text{Carry}$, update CPSR
ADD	R0-R15	R0-R15	R0-R15	-	$Rd = Rn + Rm$, Rn and Rm must not both specify the PC (R15).
	R0-R7	SP or PC	-	0-1020	$Rd = Rn + \text{imm}$, imm a multiple of four.
	SP	SP	-	0-508	$SP = SP + \text{imm}$, imm a multiple of four.
ADDS	R0-R7	R0-R7	-	0-7	$Rd = Rn + \text{imm}$, update CPSP
	R0-R7	R0-R7	-	0-255	$Rd = Rn + \text{imm}$, Rd,Rn specify the same register, update CPSP.
	R0-R7	R0-R7	R0-R7	-	$Rd = Rn + Rm$, update CPSP
RSBS	R0-R7	R0-R7	-	-	$Rd = Rn * -1$, update CPSP
SBCS	R0-R7	R0-R7	R0-R7	-	$Rd = Rn - Rm - \text{Carry}$, Rd, Rn must specify the same register, update CPSP
SUB	SP	SP	-	0-508	Immediate value must be an integer multiple of four.
SUBS	R0-R7	R0-R7	-	0-7	$Rd = Rn - \text{imm}$, update CPSP
	R0-R7	R0-R7	-	0-255	$Rd = Rn - \text{imm}$, Rd,Rn specify the same register, update CPSP.
	R0-R7	R0-R7	R0-R7	-	$Rd = Rn - Rm$, update CPSP

Bring the power of Mentimeter to PowerPoint

- ✕ Seamlessly embed your favorite Menti slide without changing windows.
- ▲ Edit and do a lot more on Mentimeter.com and sync in real-time.



Log in
to use Mentimeter

Work email

Your password

Log in

[Forgot password](#)

[Log in with SSO](#)

or



Don't have an account? [Sign up](#)

Bit Shifting

- Key operation to manage individual bits in a register.
- Can be used for fast multiplication/division with powers of 2.
 - **ASR** arithmetic-shift-right
 - **LSL** logical-shift-left
 - **LSR** logical-shift-right
 - **ROR** right-rotation
- You can only use R0-R7.
- Non-immediate operations must use the same source and destination register (i.e, `lsl r0, r0, r1`).

Logical left bit shifting

“logical shift left” → `lsl r0, r1, 4` ← num of shifts

result register ← register evaluated

g 0000 1001₁₆ (4097₁₀)

f 0001 0010₁₆ (65552₁₀)

equivalent C

statement

f = g << 4;

register mapping

f: r0

g: r1

Note: Example numbers are in hex; 4 bits -> 1 hex digit

Logical right bit shifting

“logical shift right” → `lsr r0, r1, 4` ← num of shifts

result register ← register evaluated

g 0001 0010₁₆ (65552₁₀)

f 0000 1001₁₆ (4097₁₀)

equivalent C

statement

f = g >> 4;

register mapping

f: r0

g: r1

Arithmetic right bit shifting

“arithmetic shift
right”

asr r0, r1, 4

num of shifts

result register

register evaluated

r1 0001 0010₁₆ (65552₁₀)

r0 0000 1001₁₆ (4097₁₀)

r1 FFFF EFFF₁₆ (-4097₁₀)

r0 FFFF FEFF₁₆ (-1025₁₀)

Rotate right bit shifting

Diagram illustrating the ROR instruction: `ror r0, r1, 4`

- `ror`: "rotate right"
- `r0`: result register
- `r1`: register evaluated
- `4`: num of shifts

g 0000 1001₁₆ (4097₁₀)

f 1000 0100₁₆ (268435712₁₀)

Logical Operators

- Using these instructions you can apply all the Boolean operations you learned in the first part of this module.
 - **AND**: $R_n = R_d \& R_m$
 - **EOR**: $R_n = R_d \wedge R_m$ (xor)
 - **ORR**: $R_n = R_d | R_m$
 - **BIC**: $R_n = R_d \& \sim R_m$
- update the N and Z flags according to the result for the CPSR.

Logical Operators – bit-by-bit AND

and r0, r1, r2

result register

registers evaluated

g 0000 1101 0000 0000

h 0011 1100 0000 0000

0000 1100 0000 0000

equivalent C statement

f = g & h;

register mapping

f: r0

g: r1

h: r2

AND	g	h
0	0	0
0	0	1
0	1	0
1	1	1

Logical Operators – ORR

orr r0, r1, r2

result register

registers evaluated

g 0000 1101 0000 0000

h 0011 1100 0000 0000

0011 1101 0000 0000

OR	g	h
0	0	0
1	0	1
1	1	0
1	1	1

Logical Operators – EOR

eor r0, r1, r2

result register

registers evaluated

g 0000 1101 0000 0000

h 0011 1100 0000 0000

0011 0001 0000 0000

OR	g	h
0	0	0
1	1	0
1	0	1
0	1	1

Logical Operators – MVN (MoVe and Not)

mvn r0, r1

result register

registers evaluated

r1 0000 1101 0000 0000

r0 1111 0010 1111 1111

NOT	g
1	0
0	1

Immediate Operands

- Programmers require operations with constants
 - incrementing a number (e.g. counting, array indexes)
 - initialising the value held by a program variable
- Most instructions can replace one of the input register with an immediate, i.e. an integer value.
- Because ARM MIPS aims to optimize the space of a program, immediate size varies.
- You may need to load big values in a register in multiple steps
- ... or use memory to store constants.

Adding a Constant

add r4, r4, #-1

“add immediate”
destination register

source register

constant

- Constants are prepended with the symbol #.
- They are interpreted as a signed integer.
- Encoded using two's complement.

Integer arithmetic operations in ARM

Instruction	Rd	Rn	Rm	imm	Restrictions
ADCS	R0-R7	R0-R7	R0-R7	-	$Rd = Rn + Rm + \text{Carry}$, update CPSR
ADD	R0-R15	R0-R15	R0-R15	-	$Rd = Rn + Rm$, Rn and Rm must not both specify the PC (R15).
	R0-R7	SP or PC	-	0-1020	$Rd = Rn + \text{imm}$, imm a multiple of four.
	SP	SP	-	0-508	$SP = SP + \text{imm}$, imm a multiple of four.
ADDS	R0-R7	R0-R7	-	0-7	$Rd = Rn + \text{imm}$, update CPSP
	R0-R7	R0-R7	-	0-255	$Rd = Rn + \text{imm}$, Rd,Rn specify the same register, update CPSP.
	R0-R7	R0-R7	R0-R7	-	$Rd = Rn + Rm$, update CPSP
RSBS	R0-R7	R0-R7	-	-	$Rd = Rn * -1$, update CPSP
SBCS	R0-R7	R0-R7	R0-R7	-	$Rd = Rn - Rm - \text{Carry}$, Rd, Rn must specify the same register, update CPSP
SUB	SP	SP	-	0-508	Immediate value must be an integer multiple of four.
SUBS	R0-R7	R0-R7	-	0-7	$Rd = Rn - \text{imm}$, update CPSP
	R0-R7	R0-R7	-	0-255	$Rd = Rn - \text{imm}$, Rd,Rn specify the same register, update CPSP.
	R0-R7	R0-R7	R0-R7	-	$Rd = Rn - Rm$, update CPSP

Loading large immediates

- `mov` can load values between `0x00000000` to `0x0000FFFF` to registers.
 - What about larger integers?
- `movt` can load 16-bit values into the upper 16 bits of a register (lower 16-bits remain the same).

```
mov r0, 0x1 @ r0 = 0x10000
```

```
mov r0, 0xffff @ Loading 0x7fffffff
```

```
movt r0, 0x7fff
```


Join at menti.com | use code **8623 9648**


Mentimeter

ANY QUESTIONS?



No questions from the audience!


Incoming questions will show up here so that you can answer them one by one.

👍 🗨

 CR ▾

Menti

Lecture 2 - ARM Assem...  



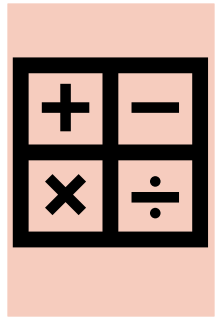
Any questions?

Questions from audience

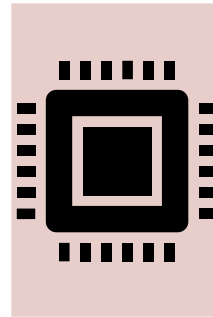
Recap Questions

- How do you load the value 0x11223344 to register r0?
- Assume $r0 = 4$, $r1 = 2$. What is the result value of r0, after the execution of the instruction *sub r0, r1*.
- Implement the following arithmetic operations: $r1 = 16 * r0 + r1 / 8$.

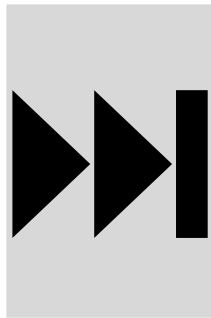
Summary



Registers and
simple arithmetic



Logical
instructions



Next

- Memory operands