Lancaster University

# ARM Assembly – Arithmetic and Logical Operators

Haris Rotsos

# Outline

Lancaster University

## Introduction to Assembly

- Instructions and operands
- Registers
- Arithmetic operators

## Today we will learn

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

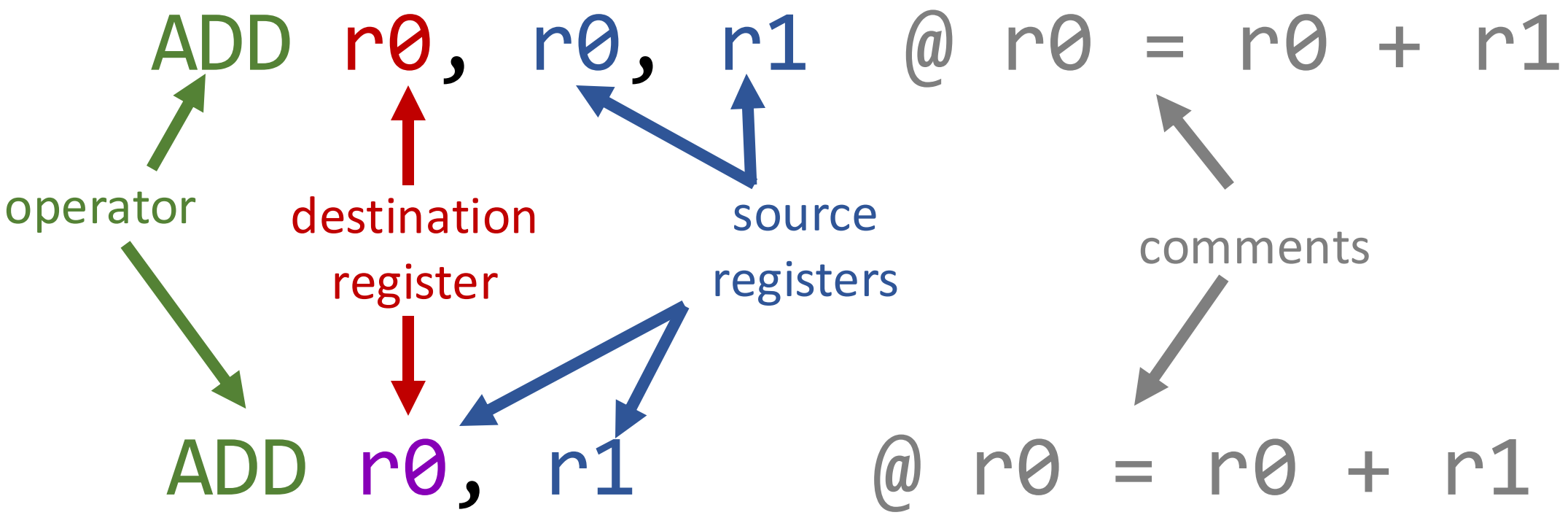
# Assembly Instruction

Lancaster University

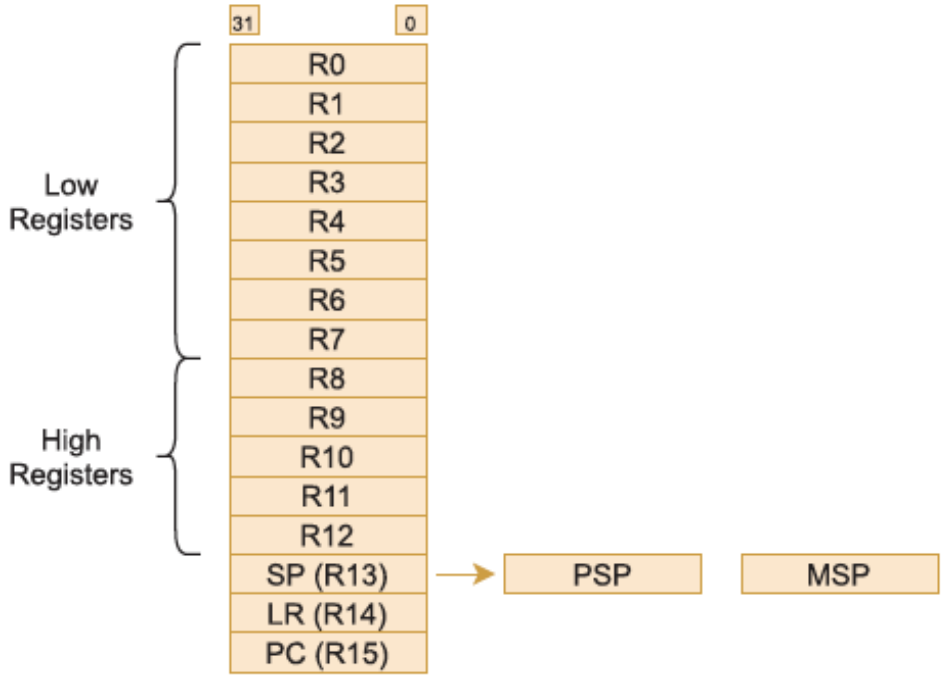
- An instruction is the most basic unit of computer processing
  - Instructions are words in the language of a computer
  - Instruction Set Architecture (ISA) is the vocabulary
- The language of the computer can be written as
  - Machine language: Computer-readable representation (that is, 0's and1's)
  - Assembly language: Human-readable representation
- We will study ARM (in detail)
- Principles are similar in all ISAs (x86, SPARC, RISC-V, …)

# Instruction Format

Lancaster University



# CPU Registers

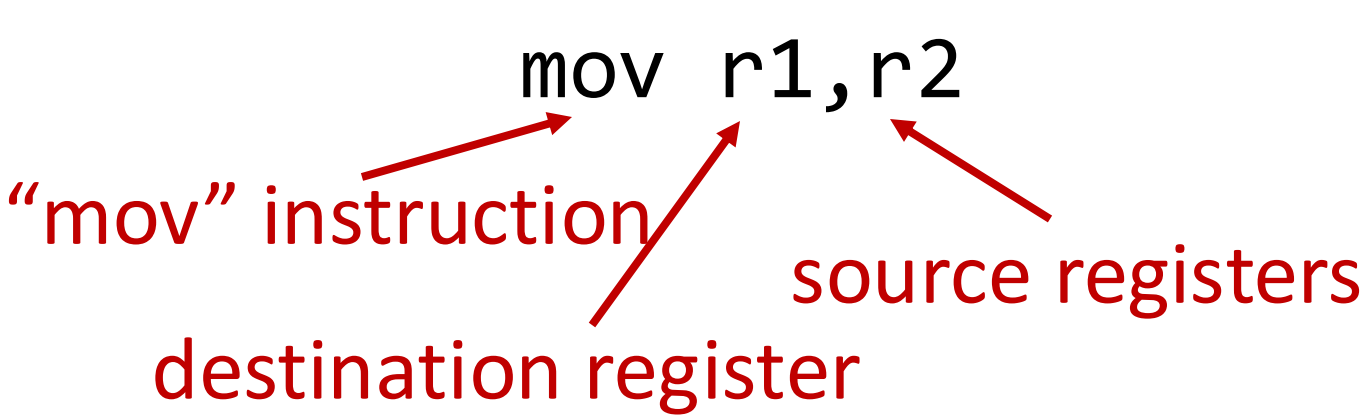


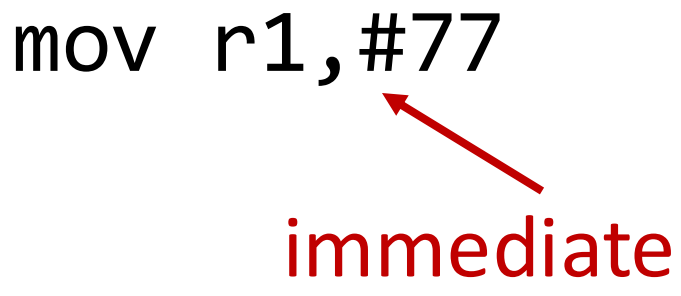

- Registers are represented with a 4-bit number.
- **R0–R7**: lower registers (bit-4 = 0)
- **R8-R12**: higher registers (bit-4=1)
  - specific instructions store data to them automatically.
- **R13** is the Stack Pointer (SP)
- **R14** is the Link Register (LR)
- **R15** is the program counter (PC)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

# Move

Lancaster University

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

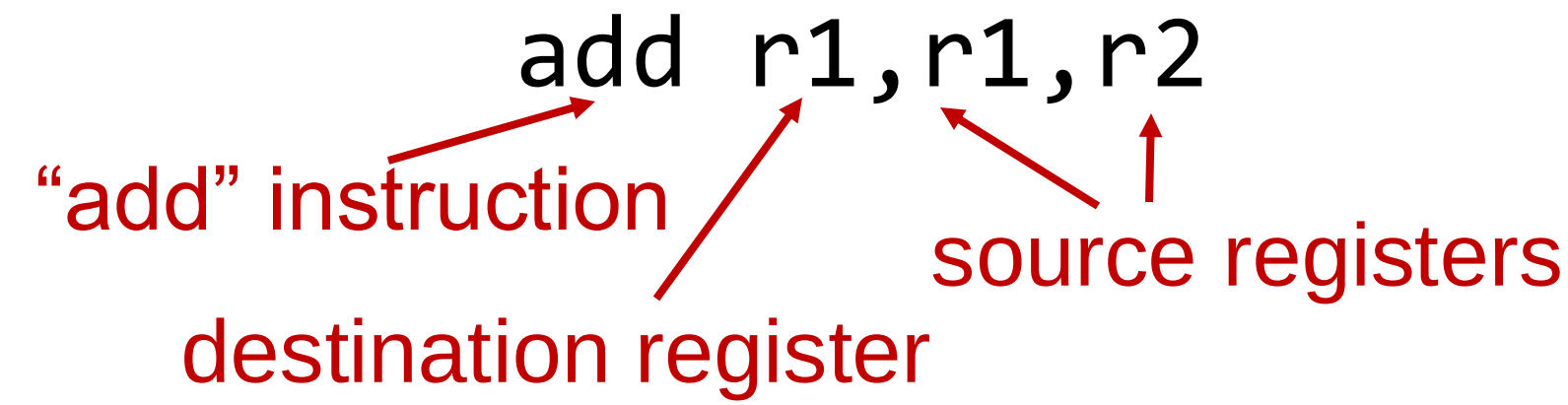



Copy value of r2 in r1

Set r1 to the value 77

# Adding



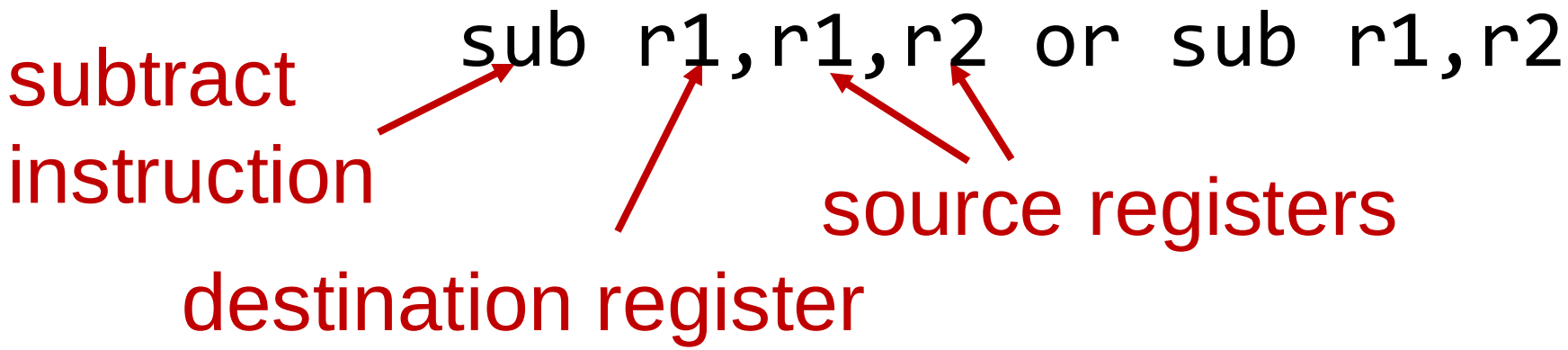

equivalent C statement

```
j = j + k;
```

register mapping

```
j: r1
k: r2
```

# Subtracting




equivalent C statement

```
j = j - k;
```

register mapping

```
j: r1
k: r2
```

# Complex Calculations

Lancaster University

```
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 !

# Number Encoding

Lancaster University

- 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

Lancaster University

- Positive number is directly converted

5 —> 0000 0101

- To obtain negative number flip bits and add 1

flip —> 1111 1010
add 1 —> 1111 1011
-5 —> 1111 1011

- To obtain the corresponding positive value flip bits again and add 1

flip —> 0000 0100
add 1 —> 0000 0101
5 —> 0000 0101

# Recording operation effects

Lancaster University

- Sometimes, more than 32-bits is needed for the outcome of an operation.
  - Carry bit.
  - Overflow.
  - Comparison.
- ARM uses a special register (CPSR) to record unusual arithmetic results.
- We will discuss this a lot when we get to loop and if statements.

# Current Program Status Register (CPSR)

Lancaster University

- 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

Lancaster University

| Instruction | Rd | Rn | Rm | imm | Restrictions |
|---|---|---|---|---|---|
| ADCS | R0-R7 | R0-R7 | R0-R7 | - | *Rd = Rn + Rm + 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 + imm*, imm a multiple of four. |
| | SP | SP | - | 0-508 | *SP = SP + imm*, imm a multiple of four. |
| ADDS | R0-R7 | R0-R7 | - | 0-7 | *Rd = Rn + imm*, update CPSP |
| | R0-R7 | R0-R7 | - | 0-255 | *Rd = Rn + 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 – 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 - imm*, update CPSP |
| | R0-R7 | R0-R7 | - | 0-255 | *Rd = Rn - imm*, Rd,Rn specify the same register, update CPSP. |
| | R0-R7 | R0-R7 | R0-R7 | - | Rd = Rn - Rm, update CPSP |

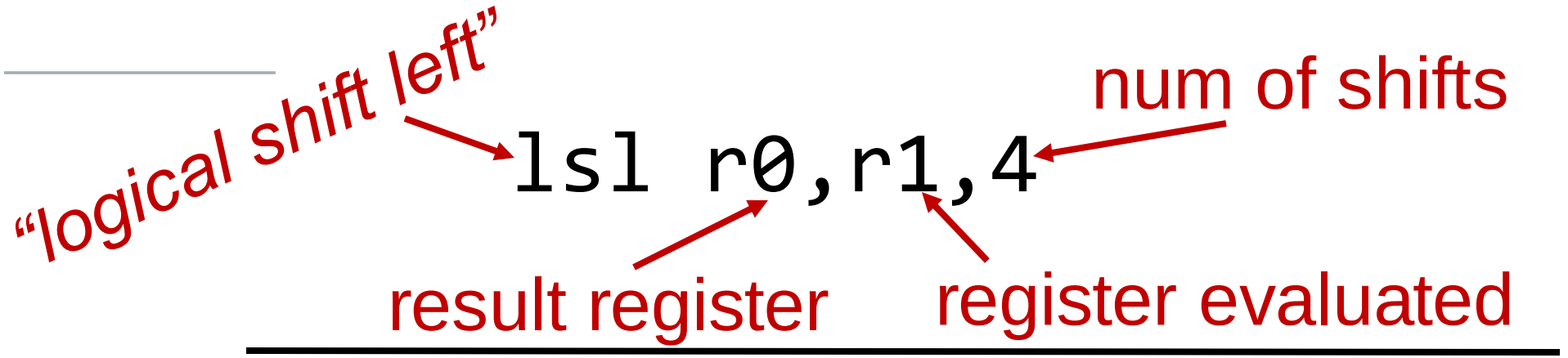
# Bit Shifting

Lancaster University

- 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



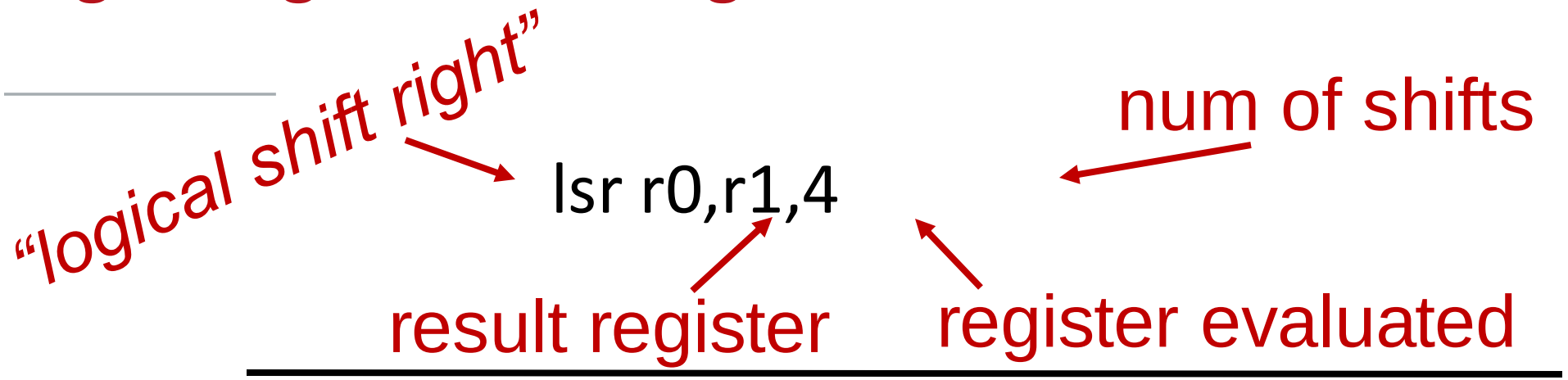

| | |
|---|---|
| g | $0000\ 1001_{16}$ $(4097_{10})$ |
| f | $0001\ 0010_{16}$ $(65552_{10})$ |

| equivalent Cstatement | register mapping |
|---|---|
| `f = g << 4;` | `f: r0`<br>`g: r1` |

Note: Example number is in hex; 4 bits -> 1 hex digit

# Logical right bit shifting

Lancaster University

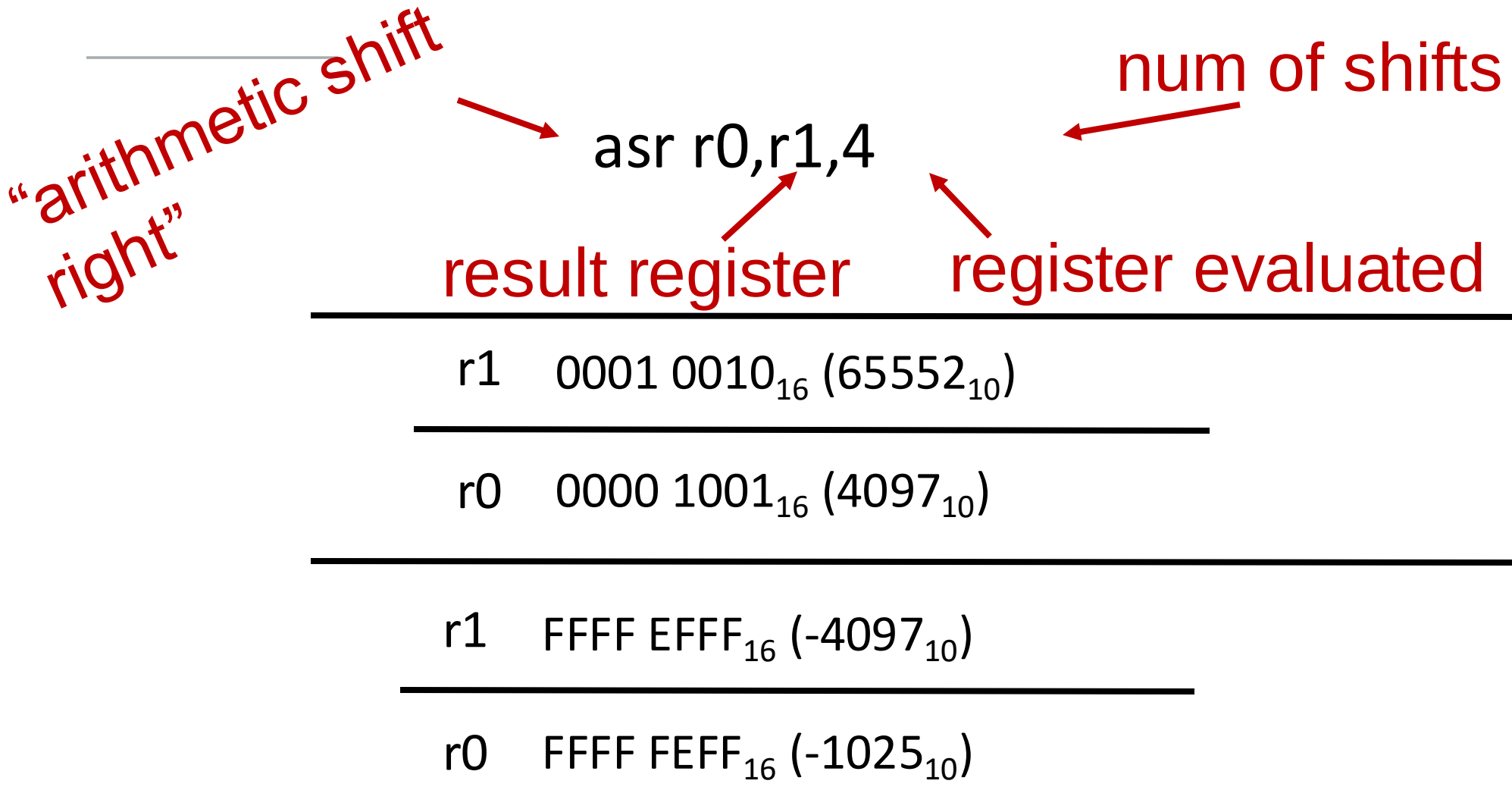



g 0001 $0010_{16}$ ($65552_{10}$)

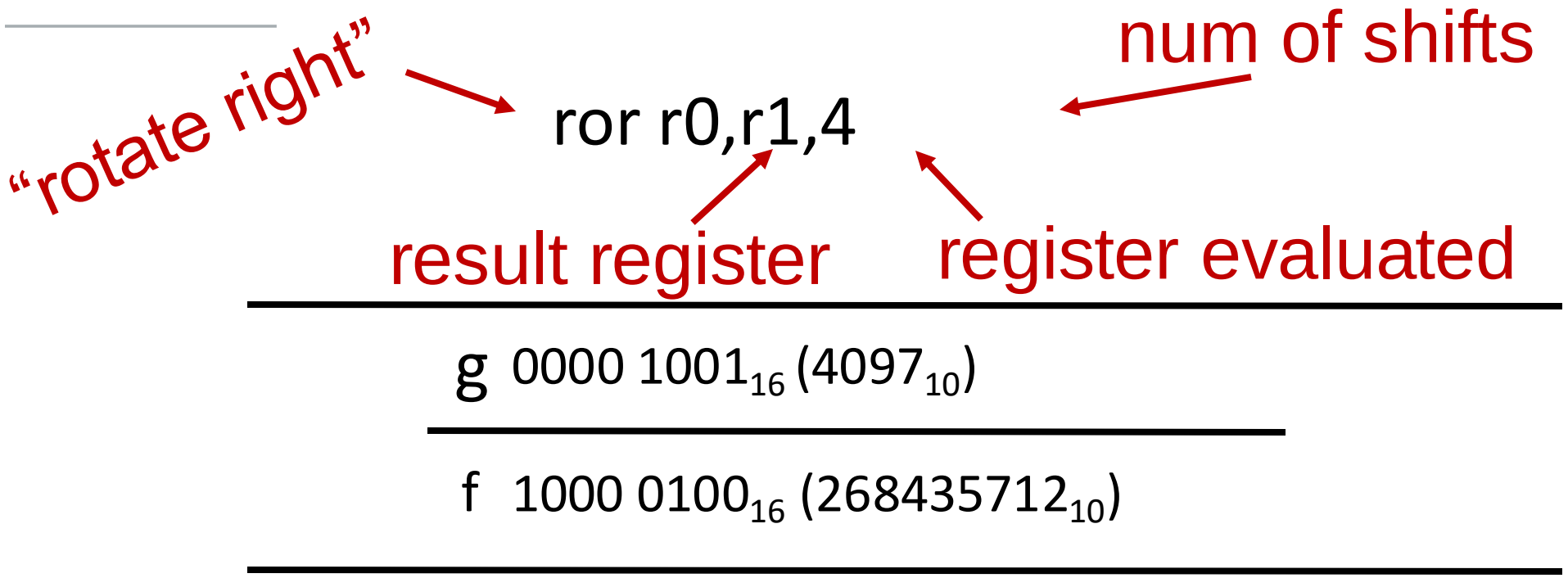
f 0000 $1001_{16}$ ($4097_{10}$)

| equivalent C statement | register mapping |
| --- | --- |
| f = g >> 4; | f: r0<br>g: r1 |

# Arithmetic right bit shifting



# Rotate right bit shifting



# Logical Operators

Lancaster University

- Using these instructions you can apply all the Boolean operations you learned in the first part of this module.
  - **AND:** Rn = Rd & Rm
  - **EOR**: Rn = Rd ^ Rm (xor)
  - **ORR**: Rn = Rd | Rm
  - **BIC:** Rn = Rd & ~Rm

- update the N and Z flags according to the result for the PCSR.

# Logical Operators – bit-by-bit AND

Lancaster University

```
and r0,r1,r2
```

result register

registers evaluated

```
g  0000 1101 0000 0000
h  0011 1100 0000 0000
-------------------------
   0000  1100  0000  0000
-------------------------
```

| AND | g | h |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

equivalent C statement

```
f = g & h;
```

register mapping

f: r0

g: r1

h: r2

# Logical Operators – ORR

Lancaster University

orr r0,r1,r2

result register

registers evaluated

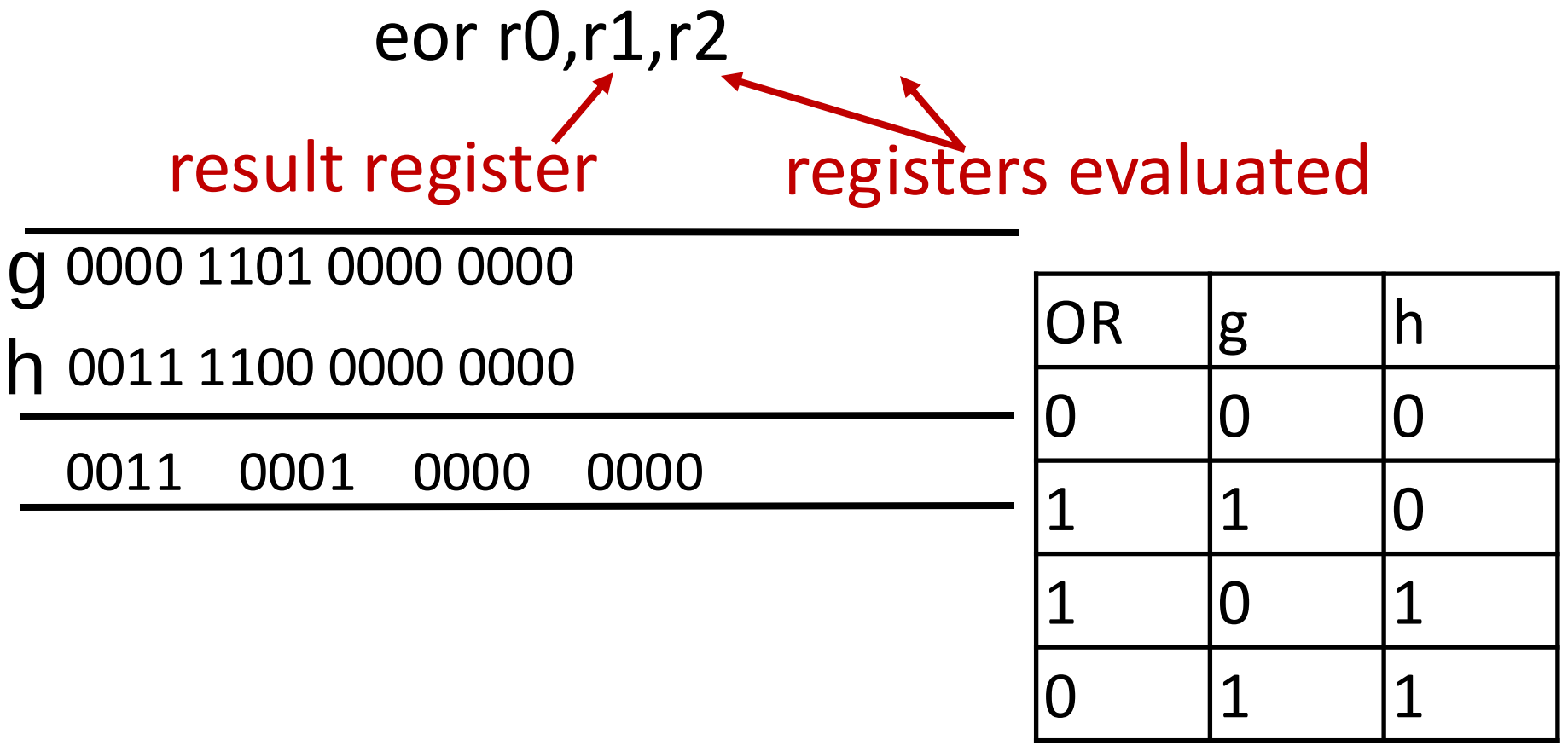
g 0000 1101 0000 0000

h 0011 1100 0000 0000

0011 1101 0000 0000

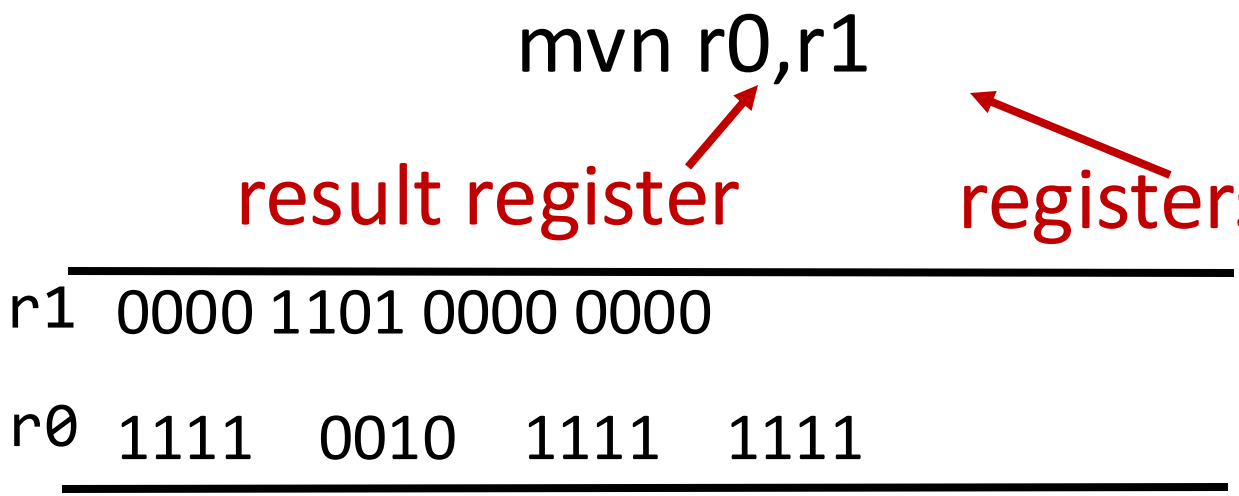
| OR | g | h |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

# Logical Operators – EOR




| OR | g | h |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |

# Logical Operators – MVN (MoVe and Not)




| NOT | g |
|---|---|
| 1 | 0 |
| 0 | 1 |

# Immediate Operands

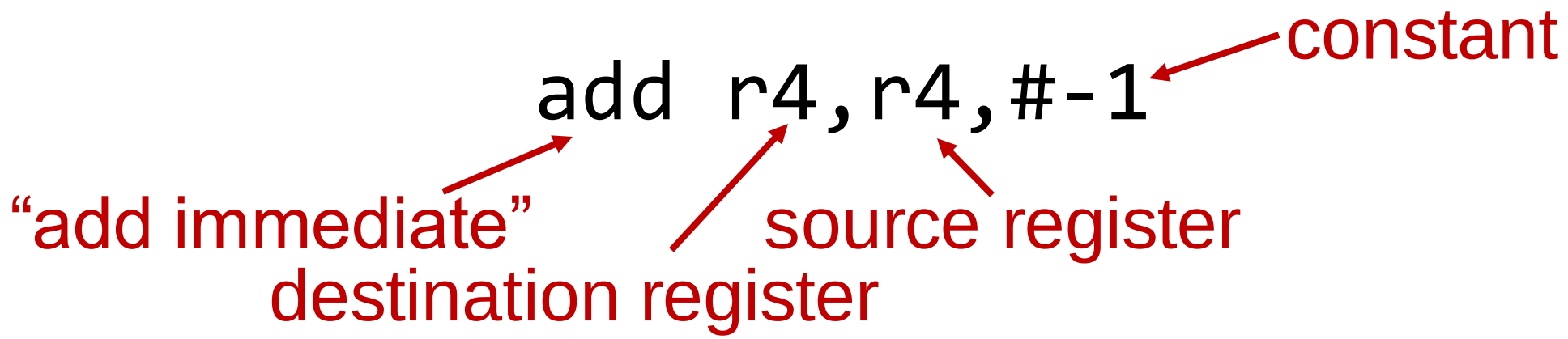
Lancaster University

- 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

Lancaster University




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

# Integer arithmetic operations in ARM

Lancaster University

| Instruction | Rd | Rn | Rm | imm | Restrictions |
|---|---|---|---|---|---|
| ADCS | R0-R7 | R0-R7 | R0-R7 | - | *Rd = Rn + Rm + 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 + imm*, imm a multiple of four. |
| | SP | SP | - | **0-508** | *SP = SP + imm*, imm a multiple of four. |
| ADDS | R0-R7 | R0-R7 | - | **0-7** | *Rd = Rn + imm*, update CPSP |
| | R0-R7 | R0-R7 | - | **0-255** | *Rd = Rn + 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 – 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 - imm*, update CPSP |
| | R0-R7 | R0-R7 | - | **0-255** | *Rd = Rn - imm*, Rd,Rn specify the same register, update CPSP. |
| | R0-R7 | R0-R7 | R0-R7 | - | Rd = Rn - Rm, update CPSP |

# Loading large immediates

Lancaster University

- `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
```

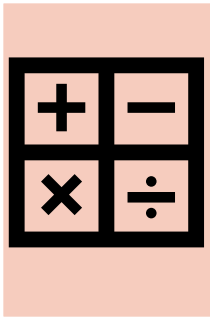
# Recap Questions

Lancaster University

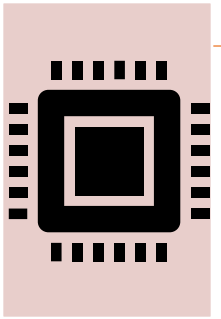
- How do you load the value 0x11223344 to register r0?
- Assume r0 = 4 , r1 = 2. What is the result value of r0, after the execution of the instruction `sub r0, r1.`
- Implement the following arithmetic operations: r1 = 16*r0 + r1/8.

# Summary

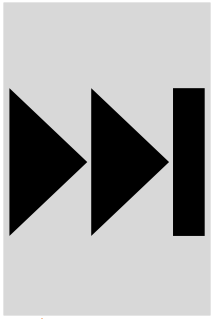
Lancaster University



Registers and simple arithmetic



Logical instructions



Next

- Memory operands