COMSM1302
Overview of Computer Architecture

Lecture 12

ARM Instruction Set

🔥 Recap

- We have designed a 4-bit CPU.

- In the labs, we have implemented our CPU.

- We have seen how the basic instructions have to be supported by hardware.

- We have programmed our first assembly code.

- We have seen how flexible assembly programming can be.

🔥 What are we missing?

- Our instruction set:
  - Is limited to 4-bit data width and just two general purpose registers.
  - Dose not support the following:
    - Conditional execution.
    - Moving data between registers.
    - Logical operations and shifts.
    - Multiplication and division.
    - Branching

ARM

- The ARM instruction set is a set of 32-bit instructions.

- There are 37 total registers in the processor.

- In user mode we can access:
    - 15 general-purpose 32-bit registers (R0 to R14)
    - Program counter (R15)
    - Current Program Status Register (CPSR)

- We will use ARM7TDMI processor.
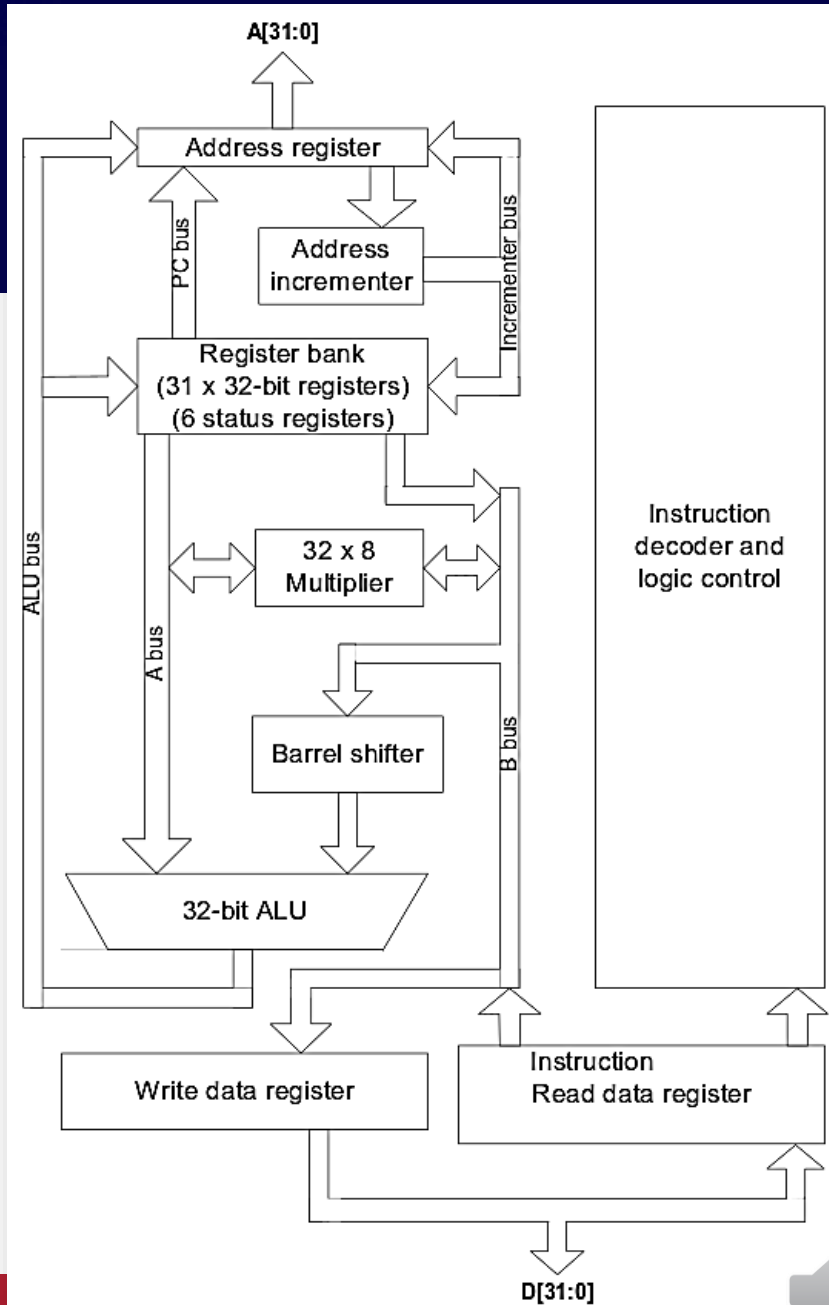
🎆 A bit of history – Acorn

Acorn Computer Ltd

Acron RISC Machine (arm)

1985

Archimedes

1987

🔥 A bit of history - Apple

Apple Computer, Inc.

1993

https://www.wired.com/2013/08/remembering-the-apple-newtons-prophetic-failure-and-lasting-ideals/

🔥 A bit of history - ARM

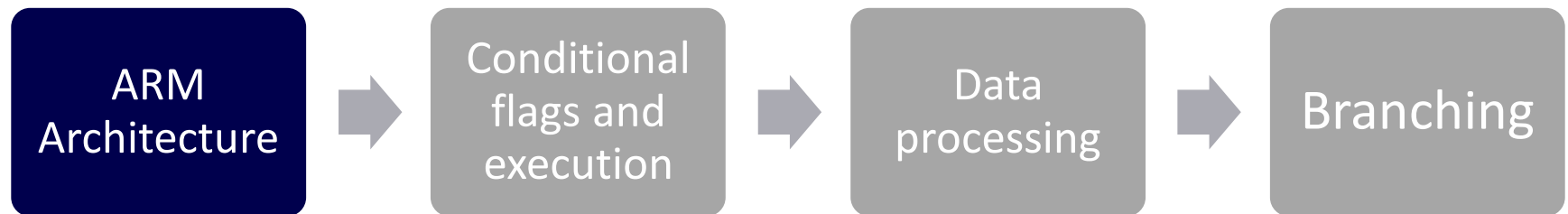Apple Computer, Inc.

Acorn Computer Ltd

VLSI Technology, Inc.

arm   1990

Advanced RISC Machine Ltd (arm)

🦅 ARM Instruction Set

ARM Architecture → **Conditional flags and execution** → **Data processing** → **Branching**

🔥 What is arm architecture? - 1

- The ARM architecture is based on *Reduced Instruction Set Computer* (RISC) principles.

- All instructions are 32 bits long.

- Most instructions execute in a single cycle.

- Every instruction can be conditionally executed.

What is arm architecture? - 2

- A load/store architecture
  - Data processing instructions act only on registers
    - Combined ALU and shifter for high speed bit manipulation.
  - Specific memory access instructions with powerful auto-indexing addressing modes.

- Instruction set extension via coprocessors

Why ARM ?

- The ARM architecture is based on a simple design.
1. A high instruction throughput.
2. An excellent real-time interrupt response.
3. A small processor die.
    1. Less material so it is cost-effective.
    2. Less transistors so low power.

University of BRISTOL

About the ARM7TDMI Core

- The ARM7TDMI core is based on the von Neumann architecture with a 32-bit data bus that carries both instructions and data.
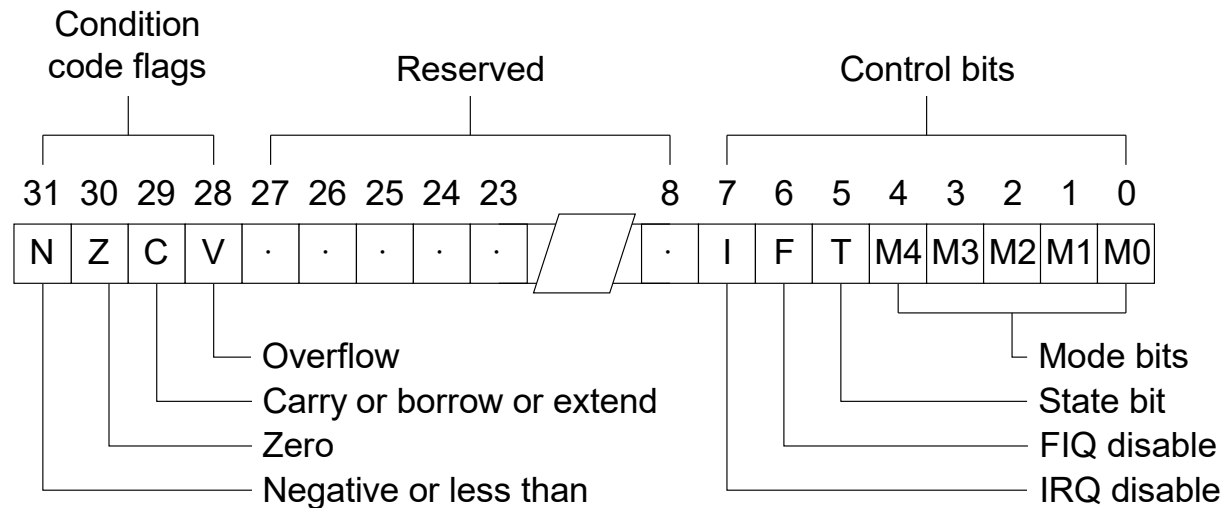
🔥 Processor Operating States

- **ARM 32-bit**, word-aligned ARM instructions are executed in this state.

- **Thumb 16-bit**, halfword-aligned Thumb instructions are executed in this state.

- https://developer.arm.com/documentation/dui0040/d/CACCIDAH
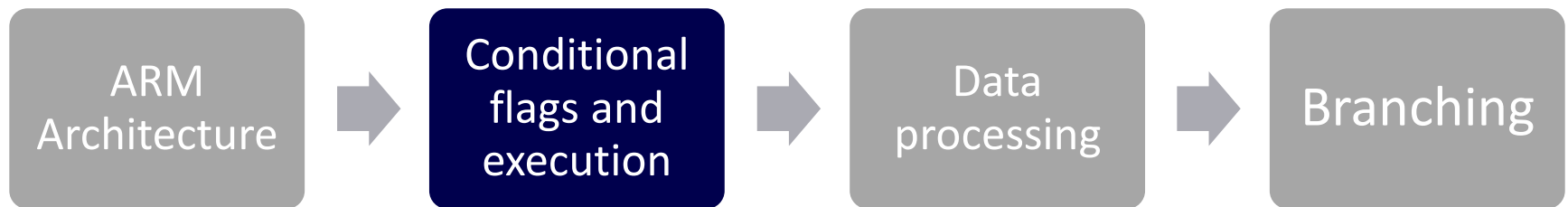
☀ The ARM-state Registers Set

1. Registers r0 to r13 are general-purpose registers used to hold either data or address values.

2. Register r14 is used as the subroutine *link Register* (LR).

3. Register r15 holds the PC.

4. By convention, r13 is used as the *Stack Pointer* (SP).

🔥 Program Status Registers

- Current Program Status Registers, CPSR.

Condition code flags				Reserved						Control bits							
31	30	29	28	27	26	25	24	23	8	7	6	5	4	3	2	1	0
N	Z	C	V	·	·	·	·	·	·	I	F	T	M4	M3	M2	M1	M0

- Overflow
- Carry or borrow or extend
- Zero
- Negative or less than

- Mode bits
- State bit
- FIQ disable
- IRQ disable

🔥 ARM Instruction Set

ARM Architecture → **Conditional flags and execution** → Data processing → Branching

University of BRISTOL

🎇 Condition Code Flags

Flag	Logical Instruction	Mathematics Instruction
Negative, N = 1	No meaning	Bit 31 of the result has been set Indicates a negative number in signed operations
Zero, Z = 1	Result is all zeroes	Result is zero
Carry, C =1	After Shift operation '1' was left in carry flag	Explained in the next slides
Overflow, V = 1	No meaning	Explained in the next slides

COMSM1302 - Computer Architecture – Anas Shrinah
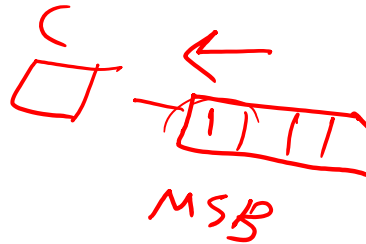
🔥 Overflow Flag - Add

- Overflow happens if the result of :
  - adding two positive numbers is a negative number
  - adding two negative numbers is a positive number
- It cannot happen when adding positive and negative numbers.
- It has no meaning if we are interpreting the data as unsigned. The ALU will still update this flag, but we can ignore it.

Overflow Flag - Sub

- Overflow happens if the result of :
  - Subtracting a positive number from a negative number is positive $(-7) - (2) = -9$
  - Subtracting a negative number from a positive number is negative $(+7) - (-2) = 9$
- It cannot happen when subtracting two positive or two negative numbers.

🔥 Carry Flag

- Carry happens when:
  - Adding two unsigned numbers and the result cannot fit in 32 bits.
  - Doing a subtraction operation and no borrowing is needed.
- It has no meaning if we are interpreting the data as signed. The ALU will still update this flag, but we can ignore it.

🔥 Conditional Execution

- All instructions can execute conditionally in ARM state.

- For example:
  - `ADD r0,r1,r2  ; r0 = r1 + r2`

- To execute this only if the zero flag is set:
  - `CMP r1,r2; Compare r1 and r2`
  - `ADD`**`EQ`**` r0,r1,r2 ;If zero flag set`

⚜ Conditional Execution - Example

- CMP r1,r2 ; Compare r1 and r2
- ADD**EQ** r0,r1,r2 ;If zero flag set
- Try this assembly code with the following values:
  1. r0 = 0, r1 = 10, and r2 = 4
  2. r0 = 0, r1 = 7, and r2 = 7
- What is the value of r0 in both cases?

🔥 Update Condition Flags

- By default, data processing operations do not affect the condition flags (apart from the comparisons where this is the only effect).

- To cause the condition flags to be updated postfix the instruction (and any condition code) with an "S".

- ADDEQ**S** or ADD**S**

- Assume:

  - r1 = #ffffffff ,r2= #1, r3 = #ff, and CPSR = 0 i.e. the condition EQ is false.

ADDS r0,r1,r2
MOVEQ r3,r0

ADD r0,r1,r2
MOVNE r3,r0

Handwritten annotations:

$Z = 0$

$Z = 1$

$Z = 1$

$r_0 = 0$

$r_3 = 0$

$r_0 = 0$

$r_3 = 0$

$Z = 0$

🔥 Condition Fields - 1

Suffix	Meaning (for cmp or subs)	Condition
EQ	Equal	Z set
NE	Not equal	Z clear
CS or HS	Unsigned higher, or same (or no borrow)	C set
CC or LO	Unsigned lower or borrow	C clear
MI	Negative	N set
PL	Positive or zero	N clear
VS	Signed overflow	V set
VC	No signed overflow	V clear

🔥 Condition Fields - 2

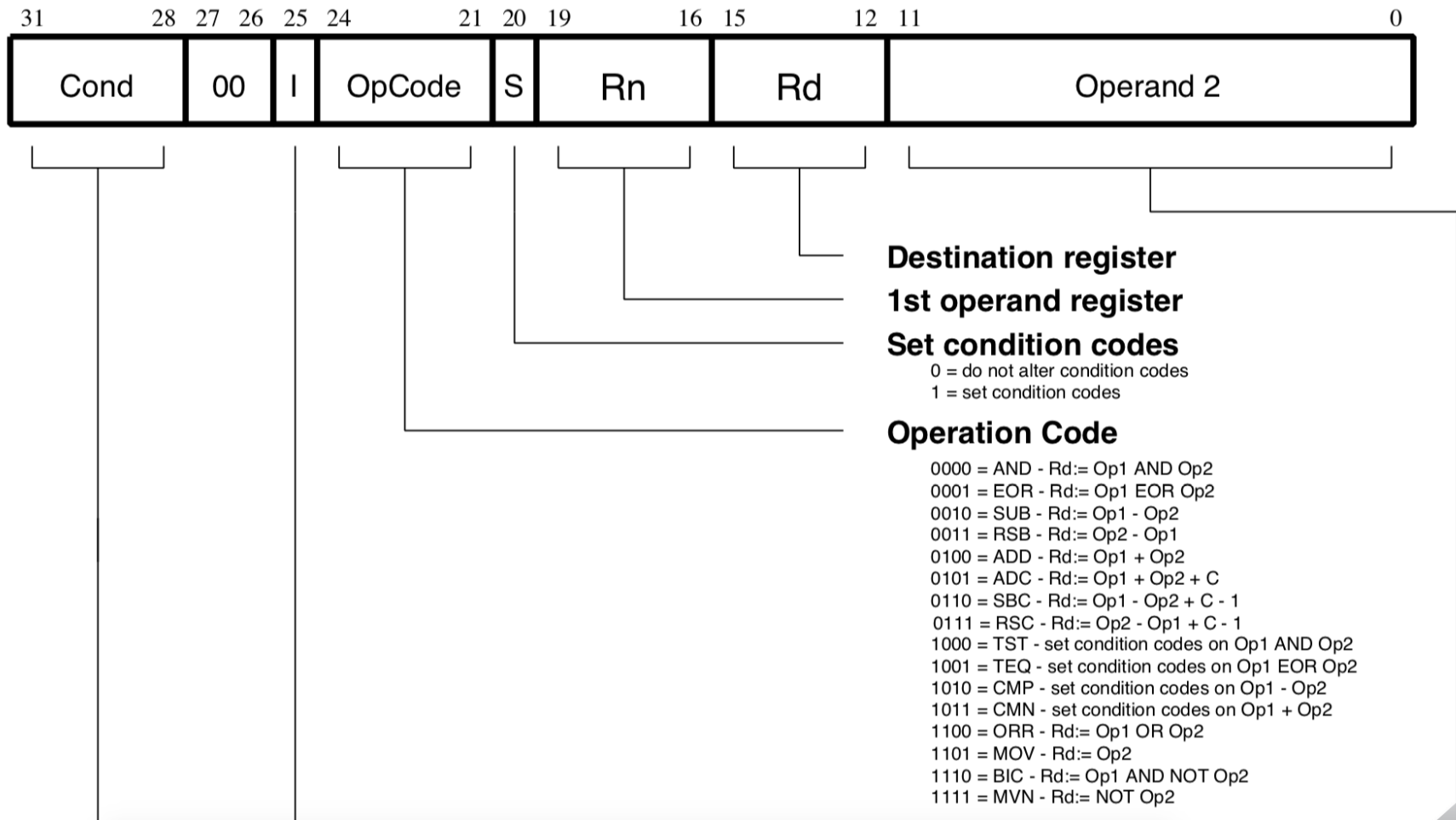| Suffix	Meaning (for cmp or subs)	Condition
HI	Unsigned higher	C set, Z clear
LS	Unsigned lower, or same	C clear, Z set
GE	Signed greater than, or equal	N=V (N and V set or N and V clear)
LT	Signed less than	N<>V (N set and V clear) or (N clear and V set)
GT	Signed greater than	Z clear, N=V (N and V set or N and V clear)
LE	Signed less than, or equal	Z set or N<>V (N set and V clear) or (N clear and V set)
AL	Always	Flag ignored

🔥 Condition fields – examples

1. `cpsr[31:28] = 0x2,` which of the conditional fields are true?

   0010

2. **Let** `r0 = 0x8000000f, r1 = 0x800000ff.` **After executing** `adds r2,r0,r1:`

   1. What is the value of `r2` ?
   2. What is the value of `cpsr[31:28]`?
   3. Which conditional fields are true ?
   4. What is the meaning of the overflow flag in this example?
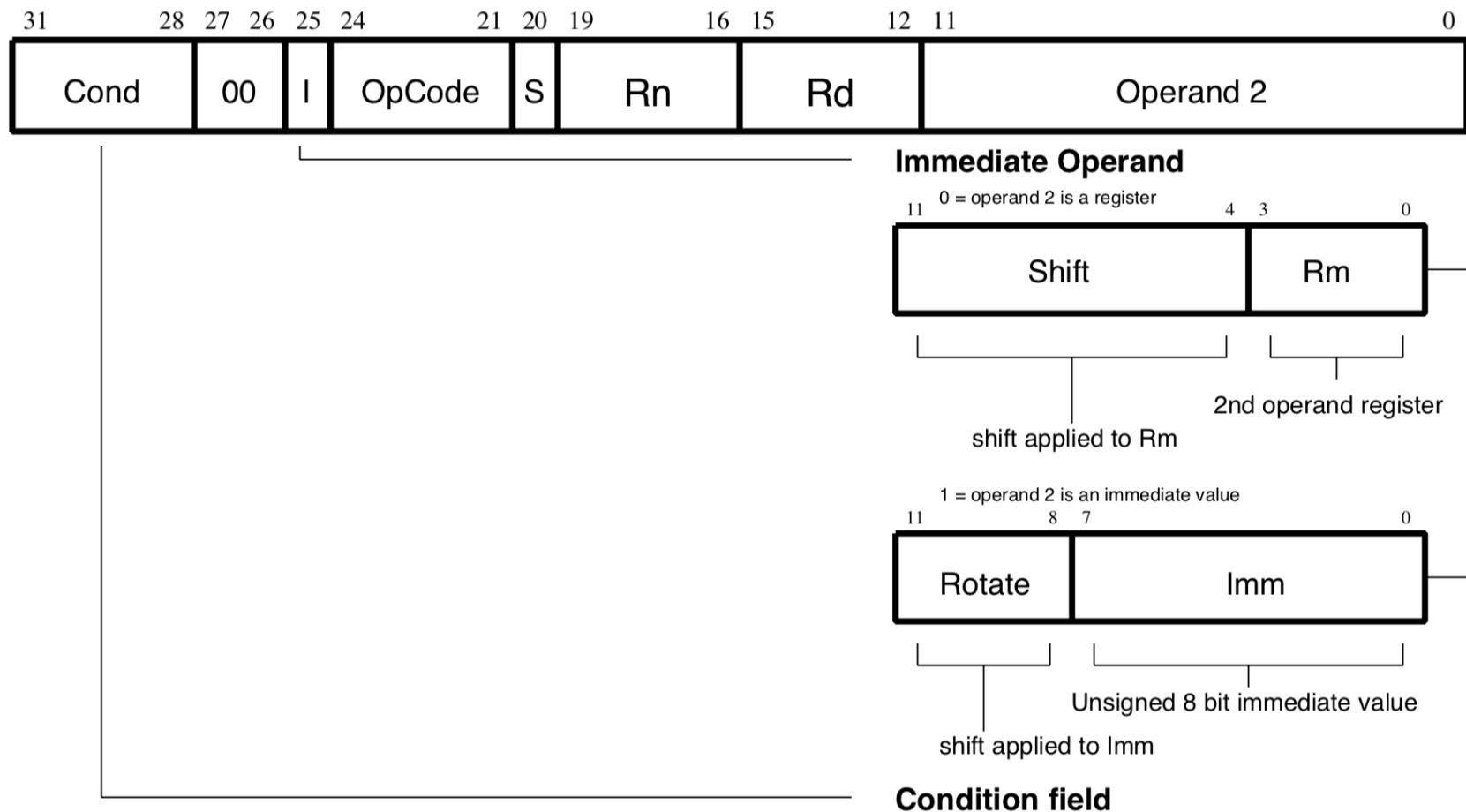
🔥 ARM Instruction Set

ARM Architecture → Conditional flags and execution → **Data processing** → Branching

🔥 Data Processing Instructions Format - 1

31 28	27 26	25	24 21	20	19 16	15 12	11 0
Cond	00	I	OpCode	S	Rn	Rd	Operand 2

Destination register

1st operand register

Set condition codes
0 = do not alter condition codes
1 = set condition codes

Operation Code
0000 = AND - Rd:= Op1 AND Op2
0001 = EOR - Rd:= Op1 EOR Op2
0010 = SUB - Rd:= Op1 - Op2
0011 = RSB - Rd:= Op2 - Op1
0100 = ADD - Rd:= Op1 + Op2
0101 = ADC - Rd:= Op1 + Op2 + C
0110 = SBC - Rd:= Op1 - Op2 + C - 1
0111 = RSC - Rd:= Op2 - Op1 + C - 1
1000 = TST - set condition codes on Op1 AND Op2
1001 = TEQ - set condition codes on Op1 EOR Op2
1010 = CMP - set condition codes on Op1 - Op2
1011 = CMN - set condition codes on Op1 + Op2
1100 = ORR - Rd:= Op1 OR Op2
1101 = MOV - Rd:= Op2
1110 = BIC - Rd:= Op1 AND NOT Op2
1111 = MVN - Rd:= NOT Op2

🔥 Data Processing Instructions Format - 2

University of BRISTOL

🔥 Data Processing Instructions

- Contains:
  - Arithmetic operations
  - Comparisons (no results - just set condition codes)
  - Logical operations
  - Data movement between registers

Arithmetic Operations - 1

- Operations are:
  - ADD   operand1 + operand2
  - ADC   operand1 + operand2 + carry
  - SUB   operand1 - operand2
  - SBC   operand1 - operand2 + carry -1
  - RSB   operand2 - operand1
  - RSC   operand2 - operand1 + carry - 1

Arithmetic Operations - 2

- Syntax:
  - \<Operation>{\<cond>}{S} Rd, Rn, Operand2

- Examples
  - ADD r0, r1, r2
  - SUBGT r3, r3, #1
  - RSBLES r4, r5, #5

🔥 Comparisons - 1

- The only effect of the comparisons is to
  - ***<u>UPDATE THE CONDITION FLAGS</u>.*** Thus no need to set S bit.

- Operations are:
  - CMP operand1 - operand2, No result
  - CMN operand1 + operand2, No result
  - TST operand1 AND operand2, No result
  - TEQ operand1 EOR operand2, No result

🔥 Comparisons - 2

- Syntax:
  - <Operation>{<cond>} Rn, Operand2

- Examples:
  - CMP   r0, r1
  - TSTEQ r2, #5

🔥 Logical Operations - 1

- Operations are:
  - AND   operand1 AND operand2
  - EOR   operand1 EOR operand2
  - ORR   operand1 OR operand2
  - BIC   operand1 AND NOT operand2 [ie bit clear]

University of BRISTOL

🔥 Logical Operations - 2

- Syntax:
  - \<Operation>{\<cond>}{S} Rd, Rn, Operand2

- Examples:
  - AND   r0, r1, r2
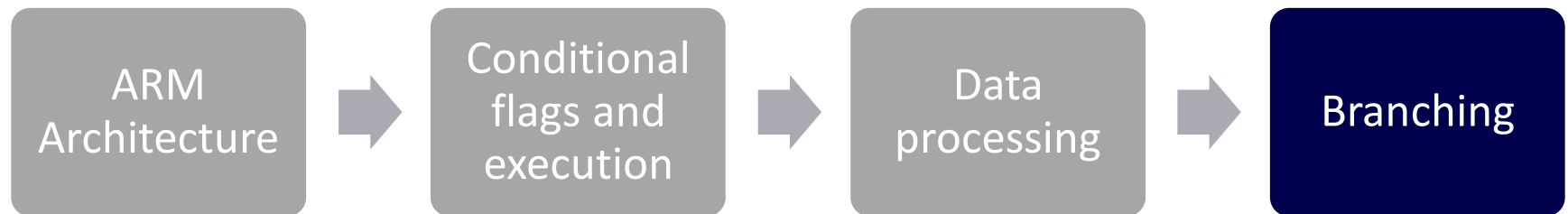 - BICEQ   r2, r3, #7
  - EORS   r1,r3,r0

🔥 Data Movement

- Operations are:
  - MOV   operand2 to the destination register
  - MVN   NOT operand2 to the destination register
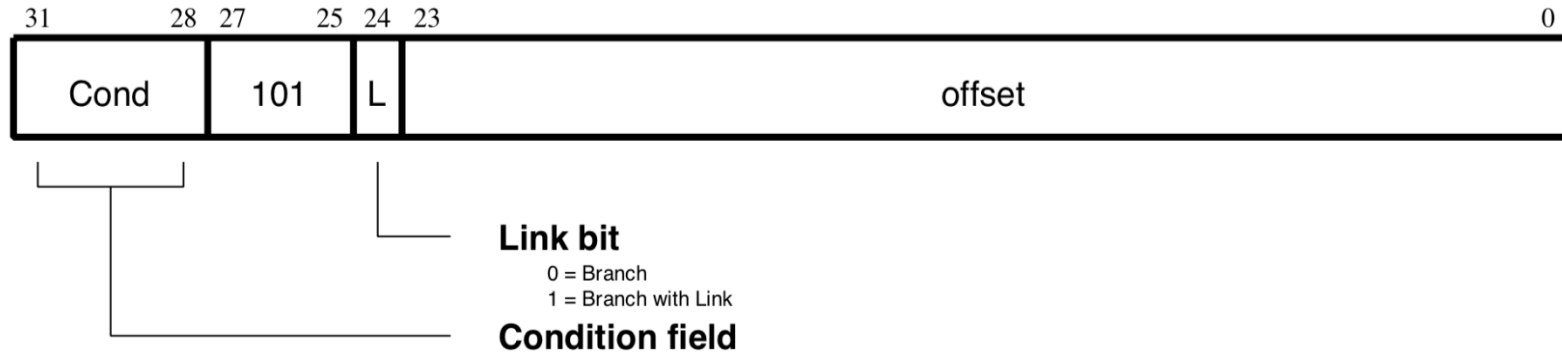 - MRS   The value of cpsr to the destination register

🔥 Data Movement

- Syntax:
  - MOV and MNV

    <Operation>{<cond>}{S} Rd, Operand2
  - MRS{cond} Rd, cpsr

- Examples:
  - MOV   r0, r1
  - MOVS   r2, #10
  - MVNEQ   r1,#0
  - MRS r0, cpsr

University of BRISTOL

🔥 ARM Instruction Set

| ARM Architecture | → | Conditional flags and execution | → | Data processing | → | **Branching** |

🔥 Branch Instructions Format

- Contains:
  - Branch
  - Branch with Link

🔥 Branch

- Syntax:
  - Branch: `B{<cond>} label`

- Examples:

```
   MOV r1,#5        R₁ = 5          Some instructions
   MOV r0,#0        R₀ = 0          ...
  _loop:
   ADD r0,#2    → ADD R₀, R₀, #2  _end:
   SUBS r1,#1      R₁ = R₁ - 1     B _end
   BNE _loop
```

Handwritten annotations:
- 16, 8, 4 (beside MOV r1,#5 / MOV r0,#0 / _loop)
- $R_1 = 5$
- $R_0 = 0$
- $R_0 = R_0 + 2$
- ADD $R_0, R_0, \#2$
- $r_0 = 2$
- $0 \times 2 = 3$
- $r_1 = 4$
- $R_1 = R_1 - 1$
- $Z = 1$
- _end:
- B _end

🔥 Branch with Link

- Syntax:
  - Branch with Link : `BL{<cond>} label`

- Operation:
  - BL writes the the address of the instruction following the BL into the link register (R14).

- Use:
  - It saves the address of the next instruction before branching to a sub-routine.
  - To return form the sub routine use `MOV pc, lr`

```
MOV r0,#5
MOV r1,#0
BL _accumulator
MOV r2,r1
B _end
_accumulator:
ADD r1,r1,r0
SUBS r0,r0,#1
BNE _accumulator
MOV PC, LR
_end:
B _end
```

$R_0 = 5$

$R_1 = 0$

$LR = add$

$R_2 = R_1$

$A -$

$R_1 = R_1 + R_0 \, / \, R_1 = 5 + 4 + 3 + 2 + 1$

$R_0 = 0$

$R_0 = 4$

$= A$

References

- ARM7TDMI processor is an implementation of ARMv4T architecture.

- References:
  - ARM Architecture Reference Manual
  - ARM7TDMI Technical Reference Manual

Summary

- Introduction to ARM architecture.

- Conditional flags and conditional execution.

- Data processing instructions .

- Branch instructions.

- Difference between architecture and implementation.