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?

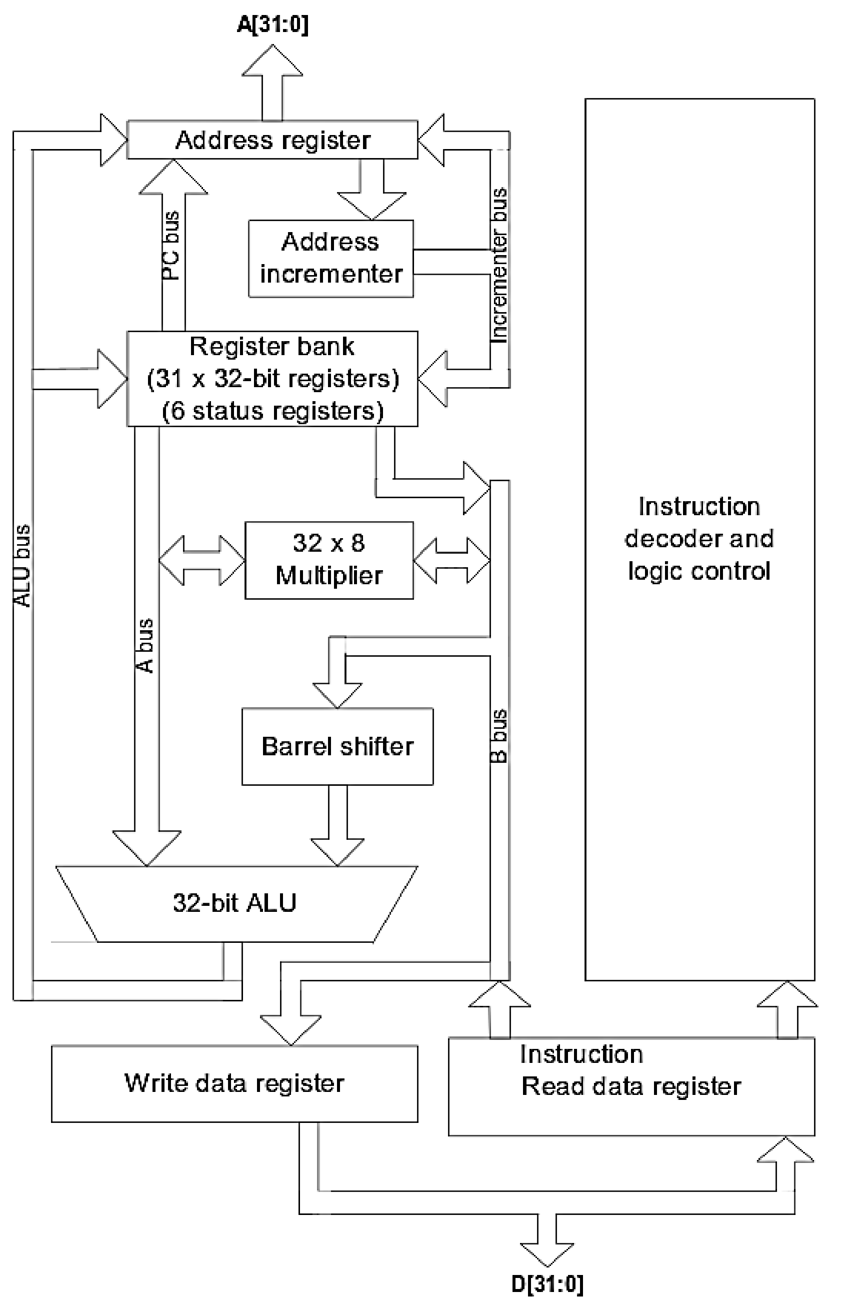
- Our instruction set:
  - Is limited to 4-bit data width and just two general purpose registers.
  - Dose not support the following:
    - Conditional execution.
    - Moving data between registers.
    - Logical operations and shifts.
    - Multiplication and division.
    - Branching

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)

16 registers

- We will use ARM7TDMI processor.



A bit of history – Acorn

Acorn Computer Ltd

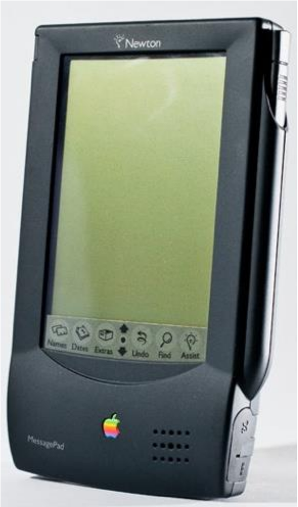
Acron RISC Machine (arm)

1985

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

Acorn Computer Ltd

Apple Computer, Inc.

VLSI Technology, Inc.

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. 模具
   1. Less material so it is cost-effective.
   2. Less transistors so low power.

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

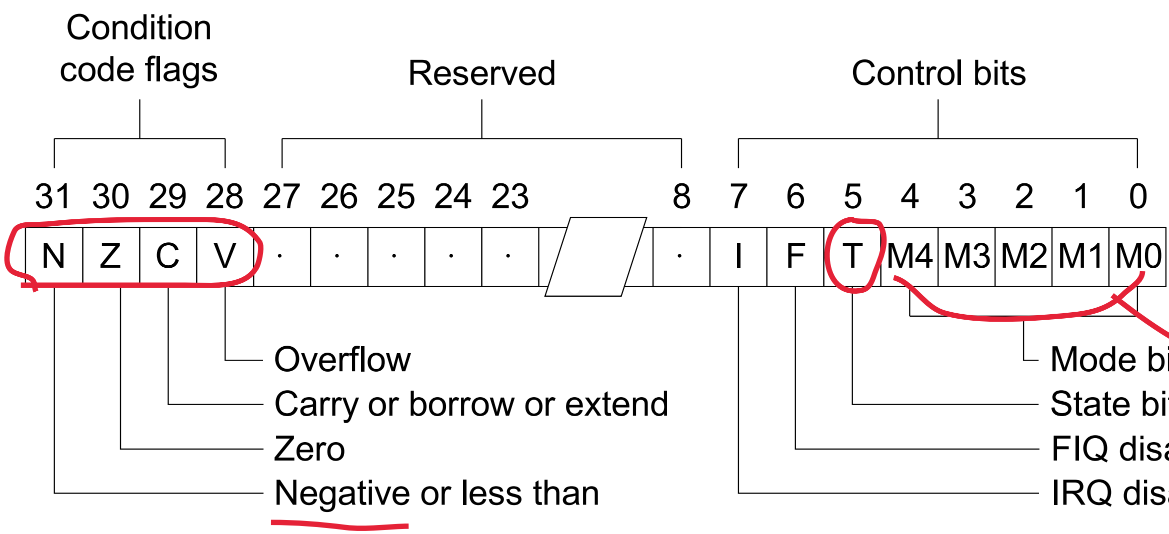
- ARM 32-bit, word-aligned ARM instructions are executed in this state.
- Thumb 16-bit, halfword-aligned Thumb instructions are executed in this state.
- https://developer.arm.com/documentation/dui0040/d/CACCIDAH

The ARM-state Registers Set

1. Registers r0 to r13 are general-purpose registers used to hold either data or address values. r0 – r12 is general actually
2. Register r14 is used as the subroutine *link Register* (LR). 14 LR
3. Register r15 holds the PC. 15 PC
4. By convention, r13 is used as the *Stack Pointer* (SP). 13 SP

Program Status Registers

- Current Program Status Registers, CPSR.



There are only 5 status in ARM, although we got 5 bits

t=0 ARM; t = 1 Thumb

We won't manipulate CPSR directly, it will be set automatically. It's important because the CPU need to check it to process other instructions. For instance, instruction ble needs to check if Z equals to Zero

ARM Instruction Set

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

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

if the carry out is 0, then carry flag will be 0, otherwise it will be 1

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
  - Subtracting a negative number from a positive number is negative
- It cannot happen when subtracting two positive or two negative numbers.

Carry Flag

- Carry happens when: **for unsigned operations**
  - 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.

1111
+0110
10101

1010
- 0001
1001

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`
  - `ADDEQ r0,r1,r2 ;If zero flag set`

if r1 == r2, Z = 0, then addeq will be execute.

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` Z=0
  2. `r0 = 0, r1 = 7, and r2 = 7` Z=1

- What is the value of r0 in both cases?

  0 14

Update Condition Flags

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

  no change after add or sub etc, but the cpu will automatically update them when executing cmp etc. adds

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

Update Condition Flags - Example

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

Z=0

Z=1

r0=1 00000000 =0

```
ADDS r0,r1,r2
MOVEQ r3,r0
```

r3=0

Z=0

r0=0

```
ADD r0,r1,r2
MOVNE r3,r0
```

r3=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

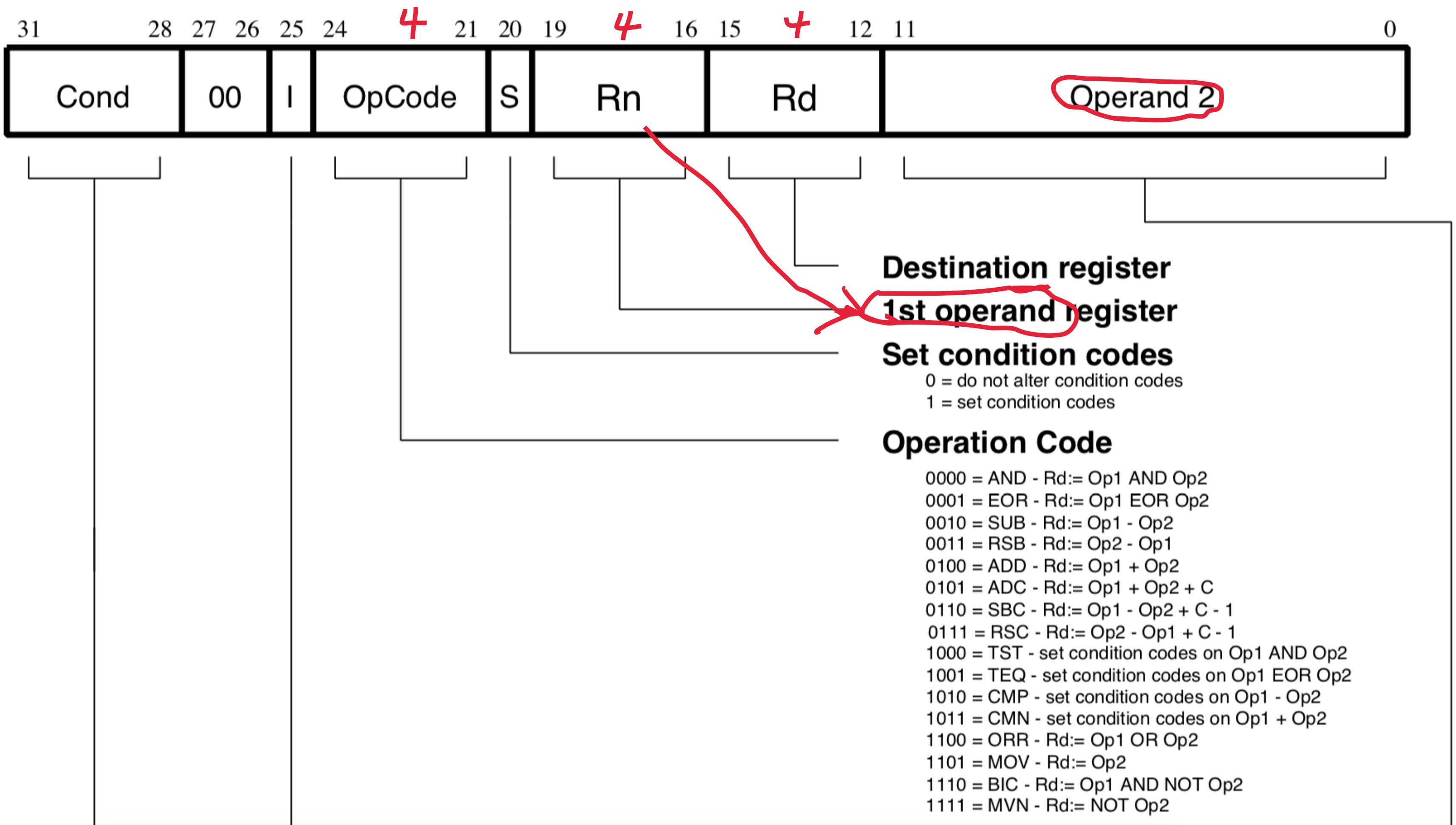
Condition fields – examples

1. `cpsr[31:28] = 0x2`, which of the conditional fields are true? C
2. Let `r0 = 0x8000000f`, `r1 = 0x800000ff`. After executing `adds r2,r0,r1`:
   1. What is the value of `r2` ? **0x0000010e**
   2. What is the value of `cpsr[31:28]`? **0x2**
   3. Which conditional fields are true ? C
   4. What is the meaning of the overflow flag in this example? **No meaning**

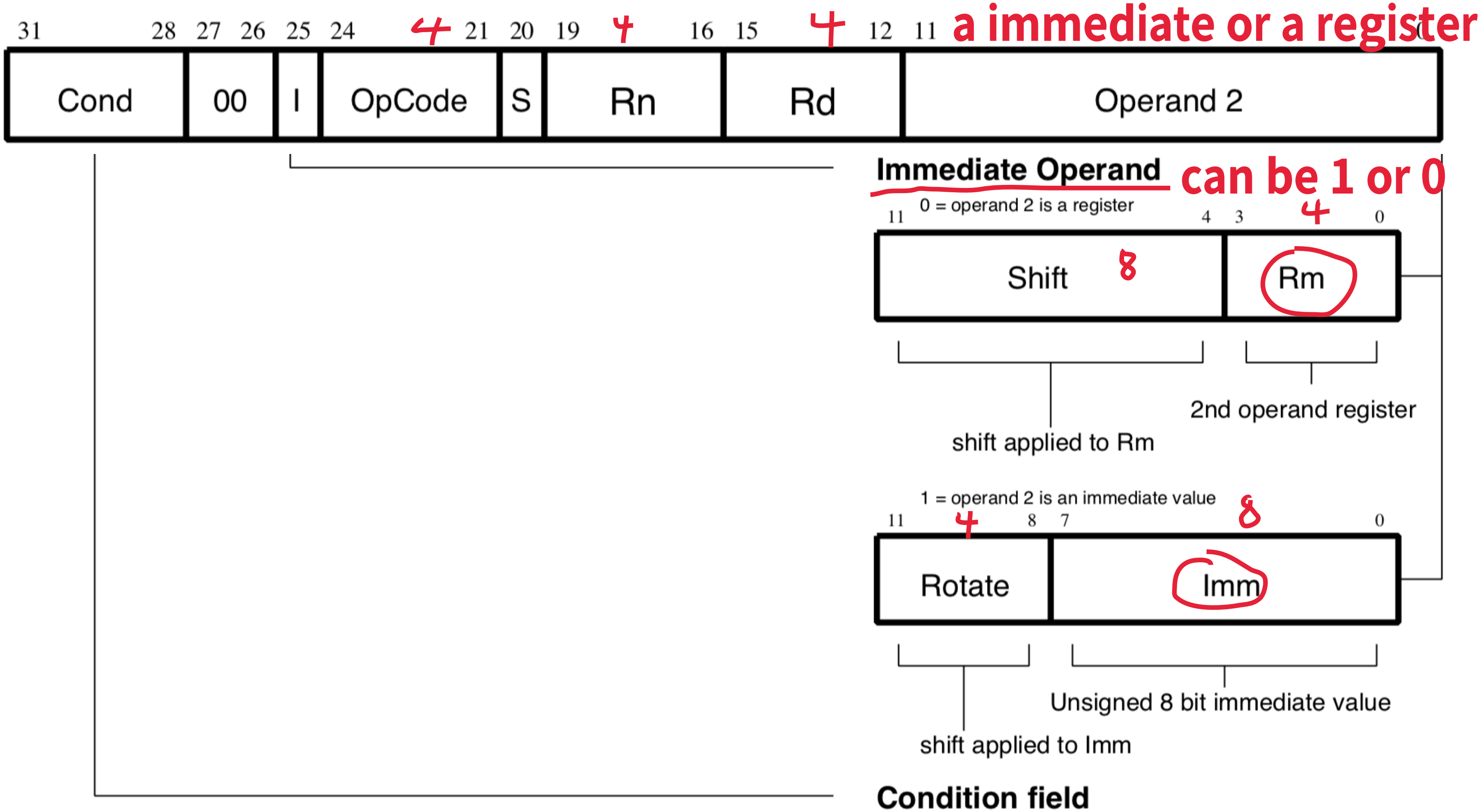
ARM Instruction Set

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

Data Processing Instructions Format - 1



Data Processing Instructions Format - 2



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   r3 = r3 - 1
  - RSBLES r4, r5, #5   r4 = 5 - r5

Comparisons - 1

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

No result

- 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]

NOT OP2 & OP1

$R_0 = 1110$

$R_1 = 0110$ NOT $R_1 = 1001$

$R_0 \wedge$ NOT $R_1 = 1000$

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

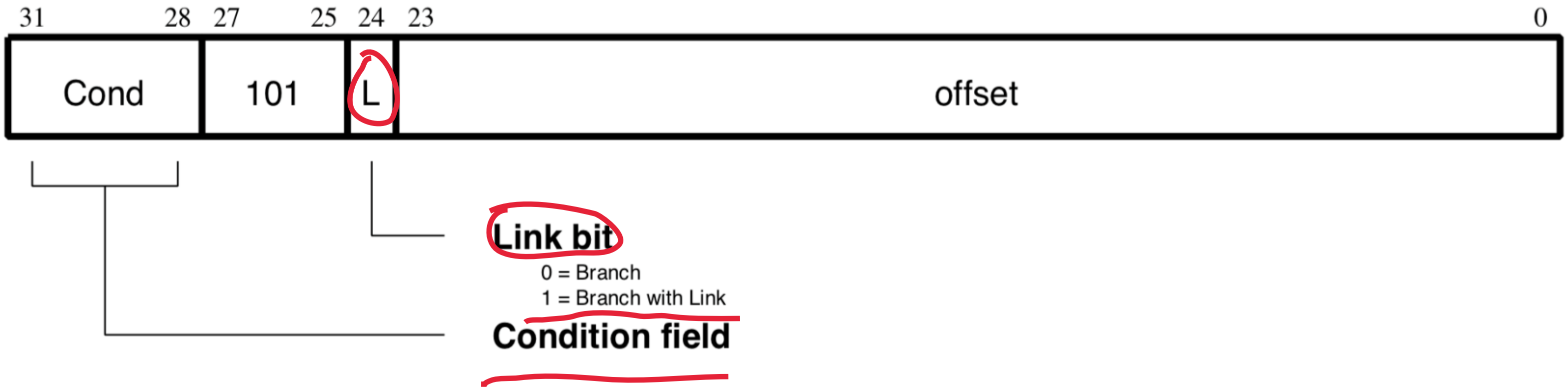
- Syntax:
  - MOV and MNV

    <Operation>{<cond>}{S} Rd, Operand2
  - MRS{cond} Rd, cpsr
- Examples:
  - MOV   r0, r1
  - MOVS   r2, #10
  - MVNEQ   r1,#0
  - MRS r0, cpsr

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:

5

```
MOV r1,#5
MOV r0,#0
_loop:
ADD r0,#2
SUBS r1,#1
BNE _loop
```

0

2

2+0=2,4,6,8,10

$R_1-1=4,3,2,1,0$

```
Some instructions
...
_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`

Branch with Link - Example

```
MOV 5r0,#5
MOV 0r1,#0
BL _accumulator
MOV r2,r1      r2 = 15
B _end
_accumulator:
ADD r1,r1,r0   5,9,12,14,15
SUBS r0,r0,#1  4,3,2,1,0
BNE _accumulator
MOV PC, LR
_end:
B _end
```

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.