# Writing ARM Assembly

Steven R. Bagley

# Hello World

```
          B main

hello     DEFB  “Hello World\n\0”
goodbye   DEFB  “Goodbye Universe\n\0”

          ALIGN

main      ADR   R0, hello    ; put address of hello string in R0
          SWI   3            ; print it out
          ADR   R0, goodbye  ; put address of goodbye string in R0
          SWI   3
          SWI   2            ; stop
```

Can now understand what this program does

```
        B main

num     DEFW  4
success DEFB  "R0 has reached the value of \0"

        ALIGN

main    LDR   R1, num
        MOV   R0, #1
next    CMP   R0, R1
        BNE   skip
        ADR   R0, success
        SWI   3
        MOV   R0,R1
        SWI   4
        MOV   R0, #10
        SWI   0
        SWI   2

skip    ADD   R0, R0, #1
        B     next
```

Lets look at another program…

Ok, a lot more going on here but it looks fairly similar at the top.

DEFW defines a 32bit (4byte) integer with the value of 4 in the bitstream…

# LDR instruction

- Mnemonic: `LDR`
- Two operands: register, and address
- Places the 32-bit value *stored* at address in the register
- Not to be confused with `ADR`
- Can also be suffixed to form `LDRB` which loads an 8-bit byte

# ADR and LDR

```
        B main

hello   DEFW  0x12345678

        ALIGN

main    ADR   R0, hello   ; put address of hello in R0
        LDR   R0, hello   ; puts the value stored at
                          ; hello in R0
```

in this case 0x12345678

# Load Store Architecture

- ARM is a load/store architecture CPU
- Simply means that memory can only be read/written by load (`LDR`) and store (`STR`) operations
- Other CPUs will let other instructions access memory as well…
- ARM separates data processing instructions from memory access

e.g. 68000, x86…

# STR instruction

- Mnemonic: `STR`
- Two operands: register, and address
- Stores the 32-bit value  in the register at the address
- Can also be suffixed to form `STRB` which loads an 8-bit byte
- Both `LDR` and `STR` can take the address in a register

```
        B main

four    DEFW  4
success DEFB  "R0 has reached the value of \0"

        ALIGN

main    LDR   R1, four
        MOV   R0, #1
next    CMP   R0, R1
        BNE   skip
        ADR   R0, success
        SWI   3
        MOV   R0,R1
        SWI   4
        MOV   R0, #10
        SWI   0
        SWI   2

skip    ADD   R0, R0, #1
        B     next
```

# Data Processing Instructions

- Only operate on registers
- Common mathematical and logical operations
- Either three or two operands…
- Usually a destination register then one or two source registers
- Last source register can optionally be an immediate value (i.e. a number)

# Data Processing Instructions

- Usually of the form
  `ADD Rd, Rn, Op2`
- Destination is the register `Rd`
- Both `Rd` and `Rn` must be registers (can be the same one)
- `Op2` can either be an immediate value or another register (which is optionally shifted)

# Data Processing Instructions

| Mnemonic | Meaning | Effect |
| --- | --- | --- |

# Data Processing Instructions

| Mnemonic | Meaning | Effect |
|---|---|---|
| AND | Logical bit-wise AND | `Rd = Rn AND Op` |
| EOR | Logical bit-wise exclusive OR | `Rd = Rn EOR Op2` |
| SUB | Subtract | `Rd = Rn - Op2` |
| RSB | Reverse Subtract | `Rd = Op2 - Rn` |
| ADD | Add | `Rd = Rn + Op2` |
| ADC | Add with carry | `Rd = Rn + Op2 + C` |
| SBC | Subtract with Carry | `Rd = Rn - Op2 + C - 1` |
| RSC | Reverse Subtract with Carry | `Rd = Op2 - Rn + C - 1` |

# Data Processing Instructions

| Mnemonic | Meaning | Effect |
|---|---|---|

TST, TEQ, CMP and CMN have no destination

# Data Processing Instructions

| Mnemonic | Meaning | Effect |
|---|---|---|
| TST | Logical bit-wise AND | Sets condition on `Rn AND Op2` |
| TEQ | Logical bit-wise exclusive OR | Sets condition on `Rn EOR Op2` |
| CMP | Subtract | Sets condition on `Rn - Op2` |
| CMN | Compare negated | Sets condition on `Rn + Op2` |
| ORR | Logical Bit-wise OR | `Rd = Rn OR Op2` |
| MOV | Move | `Rd = Op2` |
| BIC | Bit clear | `Rd = Rn AND NOT Op2` |
| MVN | Move NOT | `Rd = NOT Op2` |

TST, TEQ, CMP and CMN have no destination

# Immediate Values

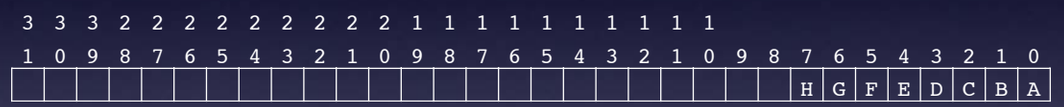
- Every instruction is 32-bits wide in ARM
- This includes the immediate value *and* the bits used to describe the instruction
- Ends up with only 12-bits being available for the immediate value
- ARM uses a clever trick to encode the immediate value

# Immediate Value

- Encoded as an 8-bit value (0-255), and a 4-bit rotation value
- Rotation shifts the bits around the 32-bits of the register

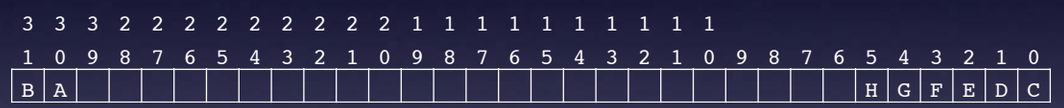
| Rotation: | 0 |
|---|---|

rotates 0 places



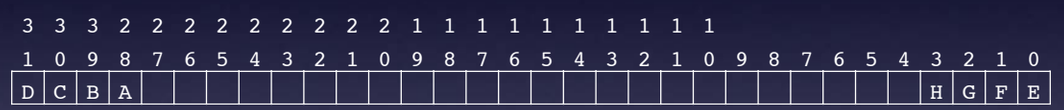
| Rotation: | 1 |
|---|---|

rotates 2 places



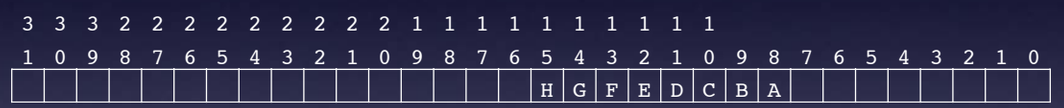
| Rotation: | 2 |
|---|---|

rotates 4 places



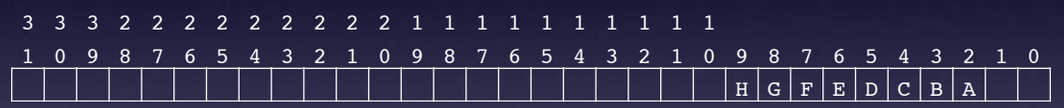
| Rotation: | 12 |
|---|---|

rotates 24 places



| Rotation: | 15 |
|---|---|

rotates 30 places



# Immediate Values

- Not possible to generate all immediate values
- But can generate a lot of the useful ones…
- Particularly if combined with `MVN`
- Again, assembler will take care of calculating rotation values

The power of letting

# Literal Pools

- Can also use `LDR` to load the value into a register from memory
- Often see ‘literal pools’ used for this
- Blocks of memory containing literal values to be load into registers
- Again, assembler can generate this automatically using a specialised form of `LDR`

```
LDR R0, =0x1234578
```

```
        B main

num     DEFW  4
success DEFB  "R0 has reached the value of \0"

        ALIGN

main    LDR   R1, num
        MOV   R0, #1
next    CMP   R0, R1
        BNE   skip
        ADR   R0, success
        SWI   3
        MOV   R0,R1
        SWI   4
        MOV   R0, #10
        SWI   0
        SWI   2

skip    ADD   R0, R0, #1
        B     next
```

Highlight Conditional branches follows the CMP

# Conditional Branch

- Saw Branch (B) instructions earlier
- These branches have extra letters in the mnemonics (BNE)
- These are *conditional* branches, branches only if the condition is true
- What condition?

# Condition Codes

| | Interpretation | CCs for execution |
|---|---|---|
| EQ | Equal / Equals Zero | $Z$ |
| NE | Not Equal | $\overline{Z}$ |
| CS/HS | Carry Set / Higher or same (unsigned) | $C$ |
| CC/LO | Carry Clear / Lower (unsigned) | $\overline{C}$ |
| MI | Minus / Negative | $N$ |
| PL | Plus / Positive or zero | $\overline{N}$ |
| VS | Overflow Set | $C \cdot \overline{Z}$ |
| VC | Overflow Clear | $\overline{C} + Z$ |

# Condition Codes

| | Interpretation | CCs for execution |
|---|---|---|
| HI | Unsigned higher | $C.\overline{Z}$ |
| LS | Unsigned lower or same | $\overline{C} + Z$ |
| GE | Greater than or Equal (signed) | N = V |
| LT | Less than (signed) | N != V |
| GT | Greater than (signed) | $\overline{Z}$ . (N = V) |
| LE | Less than or equal (signed) | Z + (N != V) |
| AL | Always | any |
| NV | Never (do not use!) | none |

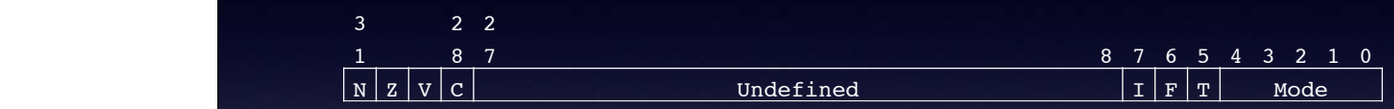
# Conditional Branch

- Condition is based on the state of flags in a special register, the `CPSR`
- Current Program Status Register
- Not part of the normal 16 registers
- Certain instructions can set flags in the `CPSR` depending on the result of the calculation…
- Requires an `S` suffix to the instruction

Originally used unused bits in R15, but has since been separated and extended…

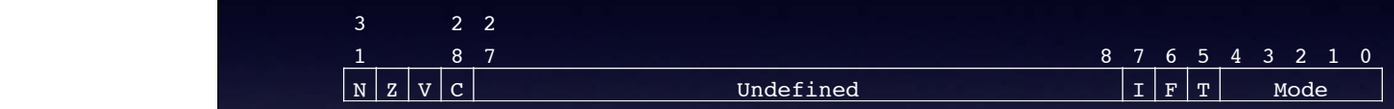
Most CPUs always update the flags, but the ARM makes the programmer do it manually

# CPSR




Carry flag, last operation caused a carry to be set

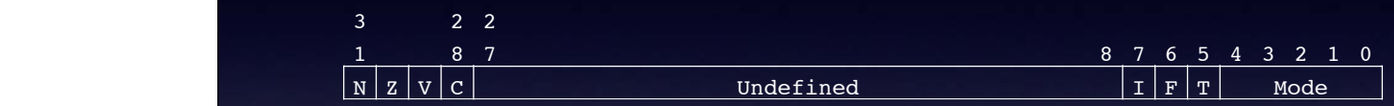
# CPSR




| | |
|---|---|
| N | Last operation yielded negative result |

Carry flag, last operation caused a carry to be set

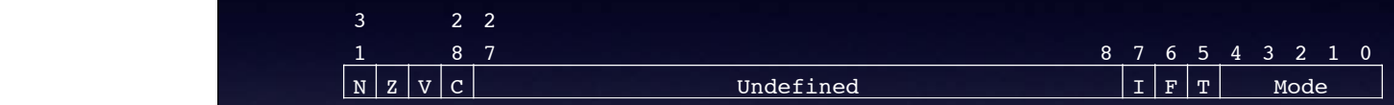
# CPSR




| | |
|---|---|
| N | Last operation yielded negative result |
| Z | Last operation yielded a zero result |

Carry flag, last operation caused a carry to be set

# CPSR




| | |
|---|---|
| N | Last operation yielded negative result |
| Z | Last operation yielded a zero result |
| V | Overflow bit… |

Carry flag, last operation caused a carry to be set

# CPSR


| | |
|---|---|
| N | Last operation yielded negative result |
| Z | Last operation yielded a zero result |
| V | Overflow bit… |
| C | Carry flag |

Carry flag, last operation caused a carry to be set