# Registers and Integers

# Registers

- Registers are physical pieces of memory adjacent to the ALU.
- Registers hold values as close to the ALU as possible.
- Values that go into the ALU are fed into it directly from registers.
- Values that are produced by the ALU go directly into registers.

# ARMv7 Registers

- In ARMv7, there are sixteen registers accessible in user-level mode: R0 - R15.
- Registers R13, R14, and R15 are designated for specific values.

- There are more registers on the processor, but:
  - They are only accessible with privilege;
  - They are only used by hardware; or,
  - They are only used when manipulating floating-point values.

# Values

- Registers hold bit strings.
- In ARMv7, these strings are 32-bits long.
- They are usually written in groups of eight bits (an octet) for readability.  That is:

  00000000 00000000 00000000 00000000

- Registers impose no meaning (e.g., variable type) on the bits.  They're just bits.

# Unsigned Integers

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

# Unsigned Integers Example

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

$$10111001_b = 128 + 32 + 16 + 8 + 1 = 185_d$$

# Range: uint8_t

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

to

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$[00000000, 11111111]_b = [0, 255]_d$

# Signed Integers: 2's-Complement

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
| --- | --- | --- | --- | --- | --- | --- | --- |
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

# Signed Integers Example

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |

$10111001_b = -128 + 32 + 16 + 8 + 1 = -71_d$

# Range: int8_t

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

to

| $2^7$ | $2^6$ | $2^5$ | $2^4$ | $2^3$ | $2^2$ | $2^1$ | $2^0$ |
|---|---|---|---|---|---|---|---|
| -128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$[10000000, 01111111]_b = [-128, 127]_d$

# Language Determines Sign

```c
uint8_t a = 255;

printf("a = %hhu\n", a);
printf("a = %hhd\n", a);

return EXIT_SUCCESS;
```

```
pi@raspberrypi:~/Documents $ ./ex
a = 255
a = -1
```

# CRITICAL POINT

- The processor does not care what the bits in the register are supposed to represent.
- To the processor, they're just 1's and 0's.
- What the bits mean is determined by the high-level language.

# Print Statement Determines Bit Interpretation

```c
uint8_t a = 92;

printf("a = %hhu\n", a);
printf("a = %hhd\n", a);
printf("a = %c\n", a);

return EXIT_SUCCESS;
```

```
pi@raspberrypi:~/Documents $ ./ex
a = 92
a = 92
a = \
```

$92_d = 01011100_b$

# Common Print Statements

- Word
  - %d = int32_t
  - %u = uint32_t
- Half-word
  - %hd = int16_t
  - %hu = uint16_t
- Half a half-word
  - %hhd = int8_t
  - %hhu = uint8_t

# Sign Extension

- ARMv7 registers are 32-bits.
- If we place an 8- or 16-bit signed integer into a 32-bit register, the sign bit should be copied to the unused bits.
- Example: int8_t = $-128_d = 10000000_b$
  - When placed in a 32-bit register without sign extension:

    00000000 00000000 00000000 10000000

  - When placed in a 32-bit register with sign extension:

    11111111 11111111 11111111 10000000

# Sign Extension: Why?

- The processor uses the same component for all ADD operations.
- Meaning: there's no "32-bit ADD", or "16-bit ADD", or "8-bit ADD".
- There's only ADD.
- This is the case for all arithmetic and logic operations.
- Using sign extension means we don't need different instructions for different sized integers.