CSE CRYPTO & SECURITY ENGINEERING Lab
암호 및 보안 공학 연구실

ARMv8-A72: ARM64 Architecture

Programming for Raspberry Pi 4B

v 1.0

Document created by:
Ji, Yong-hyeon (hacker3740@kookmin.ac.kr)

November 26, 2024

Disclaimer

The information contained in these lecture notes titled "arm64 architecture" is for educational purposes only. The contents are provided "as is" and reflect the author's views and understanding, developed from academic literature and practical experience. No warranty, express or implied, is made regarding the accuracy, adequacy, completeness, legality, reliability, or usefulness of any information. This disclaimer applies to any errors, omissions, or inaccuracies in the information.

These lecture notes are not endorsed by, directly affiliated with, maintained, authorized, or sponsored by any hardware or software company mentioned. All product and company names are the registered trademarks of their original owners. The use of any trade name or trademark is for identification and educational purposes only and does not imply any association with the trademark holder.

Any use of the information from these notes is at the user's own risk. In no event shall the author or the Cryptography and Security Engineering Lab be liable for any claims, damages, or other liabilities arising from the use or inability to use the information.

Copyright

Changelog

v1.0	2024-11-08	Initial release of the arm64 architecture lecture notes, covering all fundamental concepts and basic applications.

Table of Contents

1 ARM64 Architecture

ARM64 (AArch64) is a 64-bit architecture developed by ARM Holdings, widely used in modern mobile devices and servers.

ARM64 offers a significant register file structure, with 31 general-purpose registers and special-purpose registers such as the Program Counter (PC) and Stack Pointer (SP). These registers provide flexibility for performing low-level operations efficiently.

Key Features

- 64-bit general-purpose registers (X0-X30).
- Special-purpose registers for the stack, program control, and more.
- Optimized function calling convention.

1.1 Memory Layout in Computer Systems

In modern computer systems, memory is divided into several distinct segments that each serve a different purpose in the execution of a program. These segments include the stack, heap, and various regions that hold code and data. This section provides a mathematical description of the layout of memory in a typical system and explains the role of each memory segment.

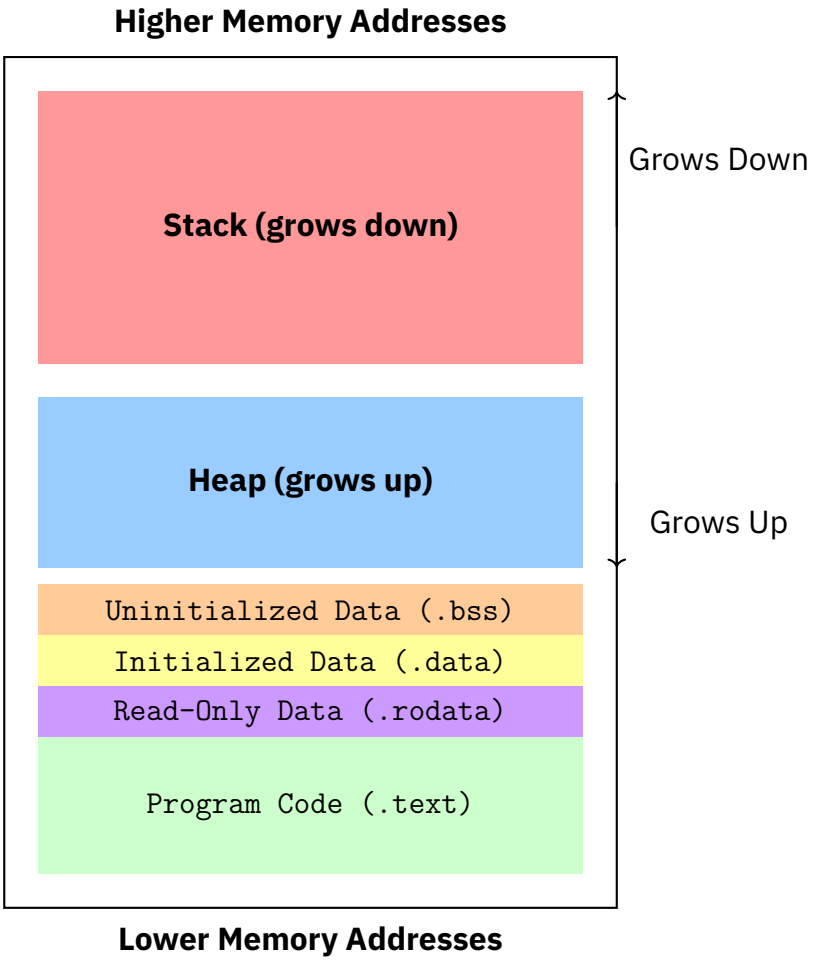
1.1.1 Memory Segments

A typical program is divided into the following memory segments:

Stack	Used for local variables, function call management, and control flow. It grows downward in memory.
Heap	Used for dynamic memory allocation (e.g., via `malloc`). It grows upward in memory.
.text	Stores the program's executable instructions (machine code).
.data	Stores initialized global and static variables.
.bss	Stores uninitialized global and static variables. This segment is initialized to zero at runtime.
.rodata	Stores read-only data, such as string literals or constant variables.

- **Stack:** Used for local variables, function call management, and control flow. It grows downward in memory.
- **Heap:** Used for dynamic memory allocation (e.g., via `malloc`). It grows upward in memory.
- **.text:** Stores the program's executable instructions (machine code).
- **.data:** Stores initialized global and static variables.
- **.bss:** Stores uninitialized global and static variables. This segment is initialized to zero at runtime.
- **.rodata:** Stores read-only data, such as string literals or constant variables.

The memory layout in a typical process can be visualized as:



1.2 Performance Metrics and Benchmarks

A computer user focuses on minimizing **response time** (or **execution time**), while a warehouse-scale operator aims to maximize **throughput**, the total work completed in a given period.[1]

[1] John L. Hennessy and David A. Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Morgan Kaufmann, 2011.

We want to relate the performance of two different computers, say, X and Y. For the phase

$$\text{“}X \text{ is faster than } Y\text{”},$$

we can define it in terms of execution times: let

- T_X denote the execution time of computer X,
- T_Y denote the execution time of computer Y.

If X is faster than Y, then:

$$T_X < T_Y$$

This inequality means that the time required for X to complete the task is less than the time required for Y.

In particular, “X is n times faster than Y”[2] means that

$$\frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n.$$

[2] The computer X is 1.5 times faster than Y means that

$$1.5 = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X}.$$

If $\text{Execution Time}_X = 10\text{s}$ and $\text{Execution Time}_Y = 15\text{s}$, TFAE:

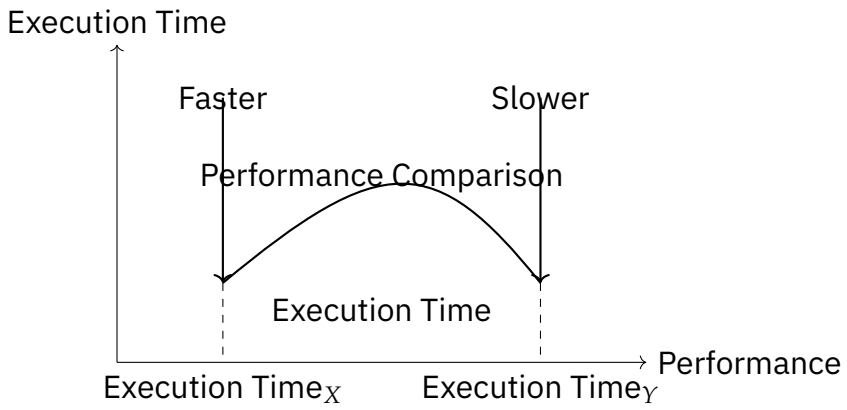
- X is 1.5 times faster than Y
- Y is about 0.76 times slower than X.

Since execution time is the reciprocal of performance, the following relationship holds:

$$n = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = \frac{1/\text{Performance}_Y}{1/\text{Performance}_X} = \frac{\text{Performance}_X}{\text{Performance}_Y}.$$

where:

- $n > 1$: Computer X is n times faster than Y.
- $n < 1$: Computer X is $1/n$ times slower than Y. In other words, computer Y is $1/n$ times faster than computer X.



1.2.1 Quantifying Performance

Understanding performance is crucial in evaluating computer systems. We use metrics such as **response time** (or *execution time*) and **throughput** to measure the efficiency of a system. The relationship between performance and execution time is inversely proportional:

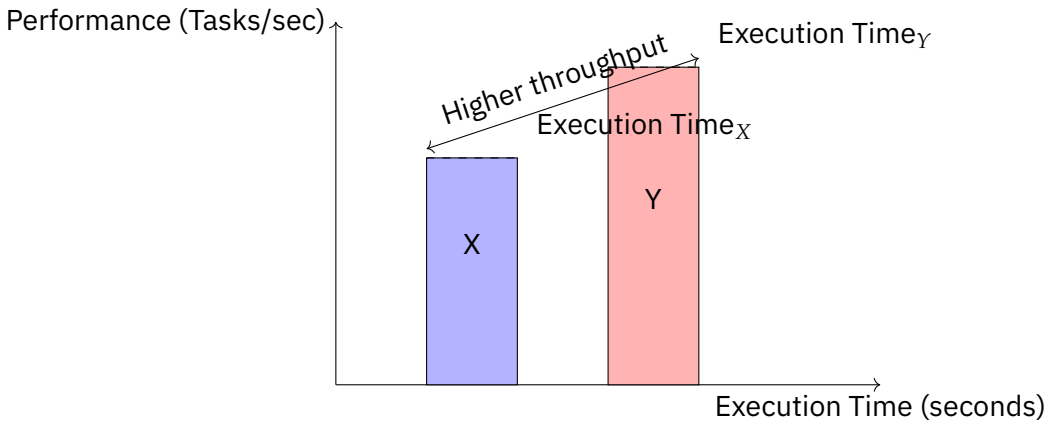
$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

This relationship allows us to compare two systems, X and Y:

$$\text{n} = \frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X}$$

1.2.2 Illustrating Performance Comparison

The following diagram represents the comparison between two systems, X and Y, in terms of their execution time and throughput:



In this example:

- System X has a shorter execution time, indicating higher performance for a given task.
- System Y has a longer execution time, leading to lower performance compared to X.

2 GNU Assembly Syntax (GAS)

GNU Assembly (GAS) follows the AT&T syntax conventions, which differ in operand order and notation from Intel syntax. Below is a structured overview of GAS syntax components and rules.

3 Load, Store and Branch Instructions

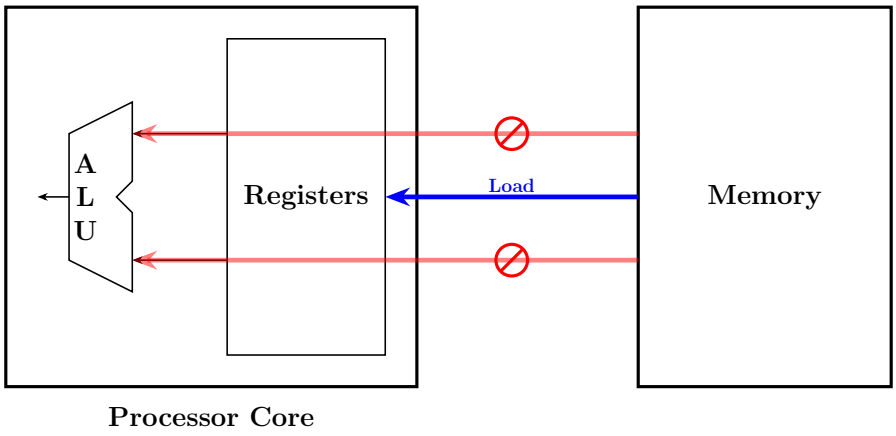


Figure 1: Loading Data from Memory

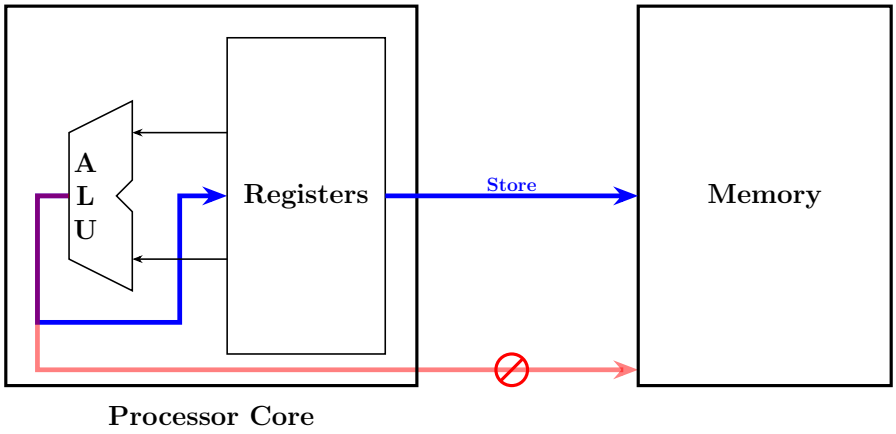


Figure 2: Storing Data to Memory

3.1 CPU components and Data paths

3.2 AArch64 User Registers

- General purpose registers
- Frame pointer
- PSTATE register
  - Negative
  - Zero
  - Carry
  - oVerflow
- Link register

- Stack pointer
- Zero register
- Program counter

3.3 Instruction Components

3.3.1 Setting and using condition flags

Example 1.

Table 1: Operation Examples

Operation	Result	N	Z	C	V
0x70000000 + 0x70000000	0xE0000000	1	0	0	1
0x80000000 + 0x80000000	0x00000000	0	1	1	1
0x90000000 + 0x90000000	0x30000000	0	0	1	1
0x00001234 - 0x00001000	0x00000234	0	0	1	0
0xFFFFFFFF - 0xFFFFFFFC	0x00000003	0	0	1	0
0x80000005 - 0x80000004	0x00000001	0	0	1	0
0x70000000 - 0xF0000000	0x80000000	1	0	0	1
0xA0000000 - 0xA0000000	0x00000000	0	1	1	0

Let

$$a = a_{31}a_{30}\cdots a_1a_0 \in \mathbb{F}_2^{32}$$
$$b = b_{31}b_{30}\cdots b_1b_0 \in \mathbb{F}_2^{32}$$
$$c = b_{31}c_{30}\cdots c_1c_0 \in \mathbb{F}_2^{32},$$

where $c = a + b \bmod 2^{32}$.

Table 2: Flag bits NZCV in PSTATE.

Name		Logical Instruction	Arithmetic Instruction
N	(Negative)	No meaning	$\mathtt{N} = c_{31}$
Z	(Zero)	Result is all zeroes	$\mathtt{Z} = \bigoplus_{i=0}^{31} c_i = 0$
C	(Carry)	-	$\mathtt{Z} = \bigoplus_{i=0}^{31} c_i = 0$
V	(oVerflow)	-	$\mathtt{Z} = \bigoplus_{i=0}^{31} c_i = 0$

3.3.2 Immediate Values

Terminology. An **immediate value** in assembly language is a *constant value* that is specified by the programmer.

Table 3: AArch64 condition modifiers.

Condition Code	Meaning	Condition Flags	Binary Encoding
EQ	Equal	$\mathtt{Z} = 1$	0000
NE	Not Equal	$\mathtt{Z} = 0$	0001
HI	Unsigned Higher	$(\mathtt{C} = 1) \wedge (\mathtt{Z} = 0)$	1000

There are two ways that immediate values can be specified in GNU ARM assembly:

(Method 1) The first way is as a literal immediate value. This can be optionally prefixed with a pound sign for clarity:

```
#<immediate|symbol>
```

(Method 2) The second way is the

```
=<immediate|symbol>
```

syntax, which can only be used with the '`ldr`' pseudo-instruction.

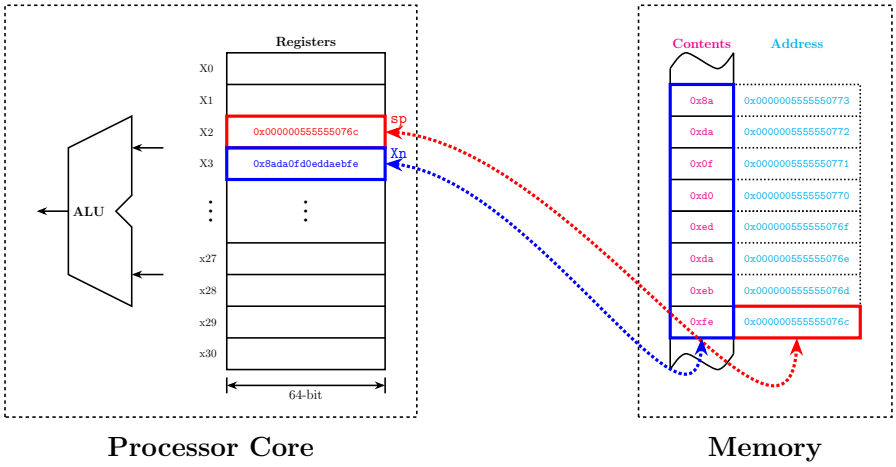
Table 4: Summary of Valid Immediate Values

Immediate Type	Bits	Description	Legal	Illegal
Arithmetic	12			
Logical	13			

3.3.3 Addressing Modes

Table 5: Load/Store Memory Addressing Modes

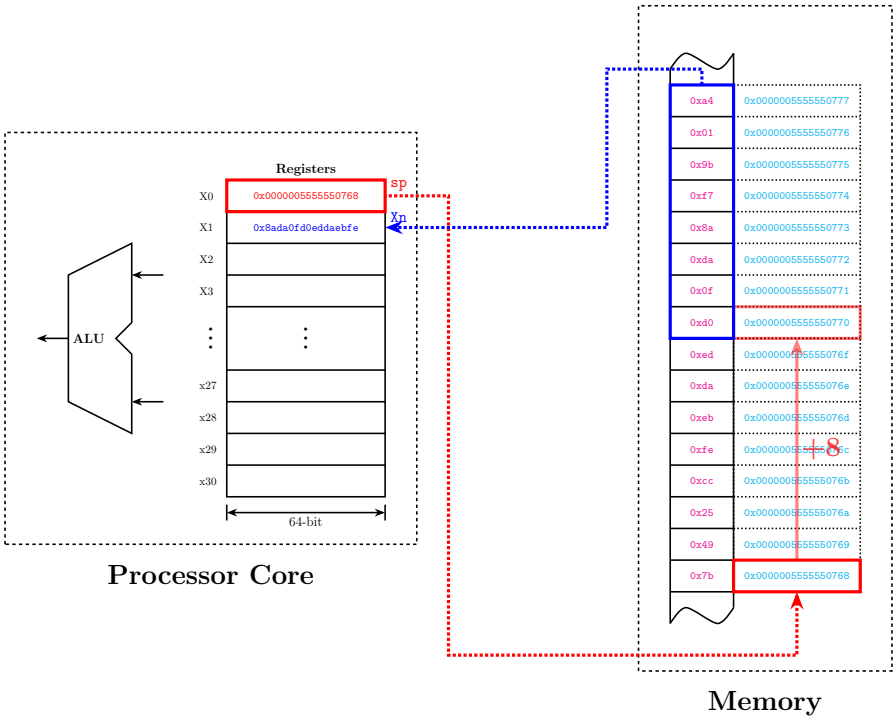
Name	Syntax	Range
Register Address	`[Xn|sp]`	
Singed Immediate Offset	`[Xn|sp, #±<imm9>]`	$\left[-2^8, 2^8\right)$
Unsinged Immediate Offset	`[Xn|sp, #<imm12>]`	$[0, 0x7ff8]$
Pre-indexed Immediate Offset	`[Xn|sp, #±<imm9>]!`	$\left[-2^8, 2^8\right)$
Post-indexed Immediate Offset	`[Xn|sp], #±<imm9>`	$\left[-2^8, 2^8\right)$
Register Offset	`[Xn|sp, Xm, (U|S)XTW]`	(or LSL #1-3)
Literal	`label`	±1 MB
Pseudo Load	`=<immediate|symbol>`	64 bits




```
ldr       x3, [x2]

x3 = memory.word[x2]

'ldr' stands for Load
to Register


str       x3, [x2]

memory.word[x2] = x3

'str' stands for Store
Register
```

Figure 3: Access the memory address containing in the register `Xn` or `sp`.

Remark. `[Xn]` or `[sp]` is just shorthand notation for `[Xn, #00]` or `[sp, #00]`, respectively.




```
ldur    x0, [x1, #0x08]

stur    x0, [x1,
#-0x08]
```

Figure 4: Signed Immediate Offset.

Table 6: Pre-index, Post-index and Pre-index with Update

Pre-index	`ldr X1, [X0, #0x08]`
	`X1 ← memory.word[X0 + 0x08]` `X0` remains unchanged
Post-index	`ldr X1, [X0], #0x08`
	`X1 ← memory.word[X0]` `X0 ← X0 + 0x08`
Pre-index with Update	`ldr X1, [X0, #0x08]!`
	`X1 ← memory.word[X0 + 0x08]` `X0← X0 + 0x08`

3.4 Load and Store Instructions

The load and store instructions allow the programmer to move data from memory to registers or from registers to memory. The load/store instructions can be grouped into the following types:

- single register,
- register pair,
- atomic.

3.4.1 Load/store single register

These instructions transfer a double-word(64-bit), single word(32-bit), half-word(16-bit), or byte(8-bit) from a register to memory or from memory to a register:

ldr Load Register
str Store Register

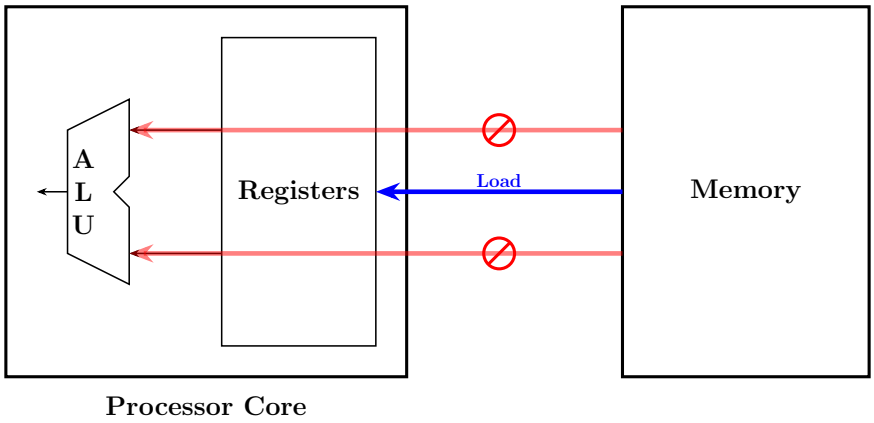


Figure 5: Loading Data from Memory

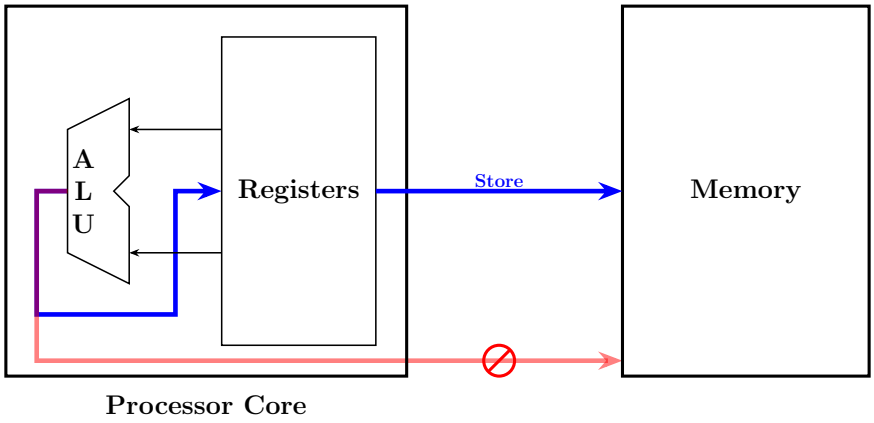


Figure 6: Storing Data to Memory

Syntax `<op>{<size>}  Rd, <addr>`

Operation

Name	Effect	Description
`ldr`	`Rd ← Mem[addr]`	Load register from memory at `addr`
`str`	`Mem[addr] ← Rd`	Store register in memory at `addr`

- `<op>` is either `ldr` or `str`.
- The optional `<size>` is one of:

  `b, h, sb, sh, sw`

  `b`: unsigned byte
  `h`: unsigned half-word
  `sb`: signed byte
  `sh`: signed half-word
  `sw`: signed word
- `str` cannot use a singed `<size>`. It also cannot use the literal addressing mode.

Example.

```
1   // Load the word (4 byte) value
2   // from Mem[x4] into w8,
3   // and set the upper four bytes of x8to zero.
4   ldr      w8, [x4]
```

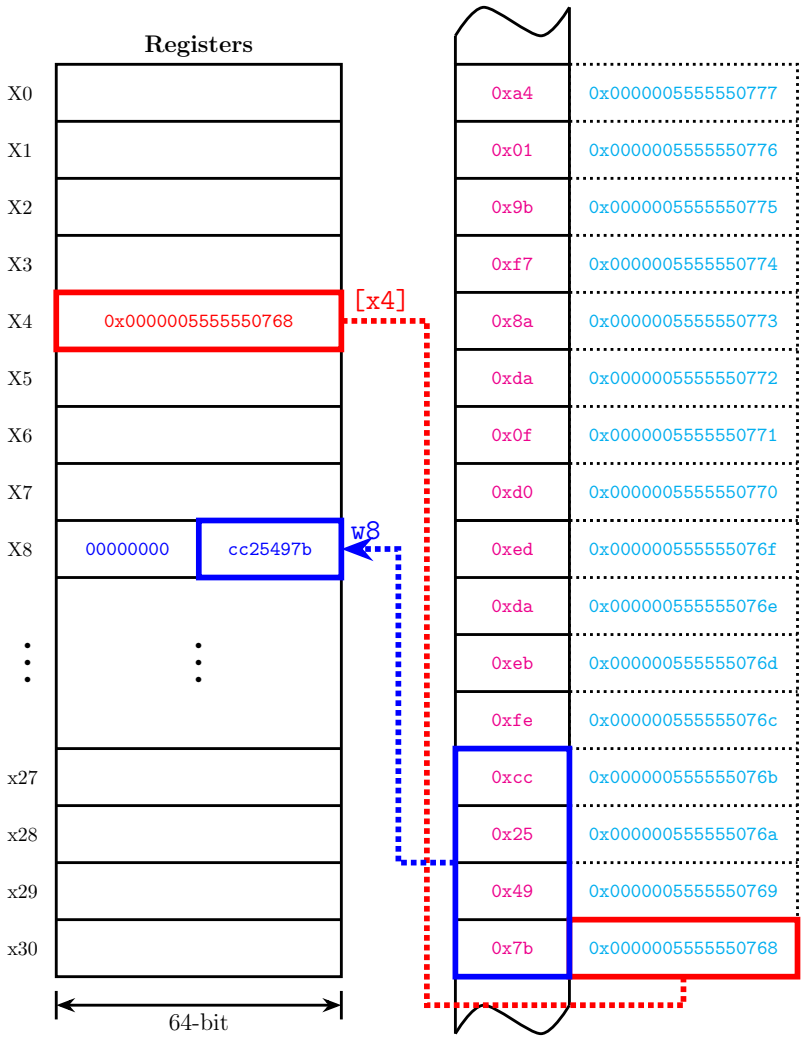


Figure 7: Load the word (4 byte) value from `Mem[x4]` into `w8`, and set the upper four bytes of `x8` to zero.

```
// Store the least-significant byte
// from register x12 into Mem[x2].
strb      x12, [x2]
```

```
// Load the double-word (8 byte) value
// from Mem[x3 + 7] into x5. Then set x3 = x3 + 7:
ldr       x5, [x3, #7]!
```

```
// Store the half-word (2 byte) value
// in w9 to Mem[x6]. Then set x6 = x6 + 7:
strh      w9, [x6], #7
```

```
// Load the half-word value
// from Mem[x0 + 8] into x5 and sign extend it:
ldrsh     x5, [x0, 8]
```

```
// Store the least significant byte
// in w1 at Mem[x9]:
strb      w1, [x9]
```

3.4.2 Load/store single register (unscaled)

These instructions are the same as Load/Store Single Register, except that they only use an unscaled, signed addressing mode with an offset range of $[-256, 256]$.

Programmers rarely need to write `ldur` or `stur` explicitly. The programmer can just use `ldr` or `str`, and the assembler will almost always automatically convert them to `ldur` or `stur` when appropriate.

ldur Load Register (Unscaled)

stur Store Register (Unscaled)

Syntax `<op>{<size>}  Rd, [Xn, #imm9]`

Operation

Name	Effect	Description
`ldur`	`Rd` ← `Mem[addr]`	Load register from memory at `addr`
`stur`	`Mem[addr]` ← `Rd`	Store register in memory at `addr`

Example.

```
// Load the byte value from Mem[x5 + 255].
// Sign extend it and store the value in x4:
ldursb      x4, [x5, #255]
```

```
// Store the double-word value in x1 to Mem[x2 - 256]:
stur      x1, [x2, #-256]
```

3.4.3 Load/store pair

These instructions are used to store or load two registers at a time. This can be useful for moving registers onto the stack or for copying data. These two instructions are particularly useful for transferring data in a load-store architecture because each instruction can move twice as much information as the `ldr` and `str` instructions.

ldp Load Pair

stp Store Pair

Syntax `<op>{<size>}  Rt, Rt2, <addr>`

- `<op>` is either `ldp` or `stp`.
- The optional `<size>` is optionally `sw` for signed words.
- `<addr>` is 7 bits Pre-indexed, Post-indexed, or Signed immediate.
- Signed immediate
  `Xt` range: $[-0x200, 0x1f8]$.
  `Wt` range: $[-0x100, 0xfc]$.

Operation

Name	Effect	Description
`ldp`	`Rt ← Mem[addr]` `Rt2 ← Mem[addr + size(Rt)]`	Load register pair from memory at `addr` where `sizeof(Rt)` is 4 for `Wt` registers and 8 for `Xt` registers
`stur`	`Mem[addr] ← Rd` `Mem[addr + size(Rt)] ← Rt2`	Store register pair in memory at `addr`

Example.

```
1       // Load the byte value from Mem[x5 + 255].
2       // Sign extend it and store the value in x4:
3       ldursb      x4, [x5, #255]
```

```
1       // Store the double-word value in x1 to Mem[x2 - 256]:
2       stur     x1, [x2, #-256]
```

3.4.4 Summary

3.5 Branch instructions

Branch instructions allow the programmer to change the address of the next instruction to be executed. They are used to implement loops, if-then structures, subroutines, and other flow control structures. There are five instructions related to branching:

- Branch,
- Branch to Register,
- Branch and Link (subroutine call),
- Compare and Branch, and
- Form program-counter-relative Address.

3.5.1 Branch

4 Data Processing and Other Instructions

4.1 Arithmetic Operations

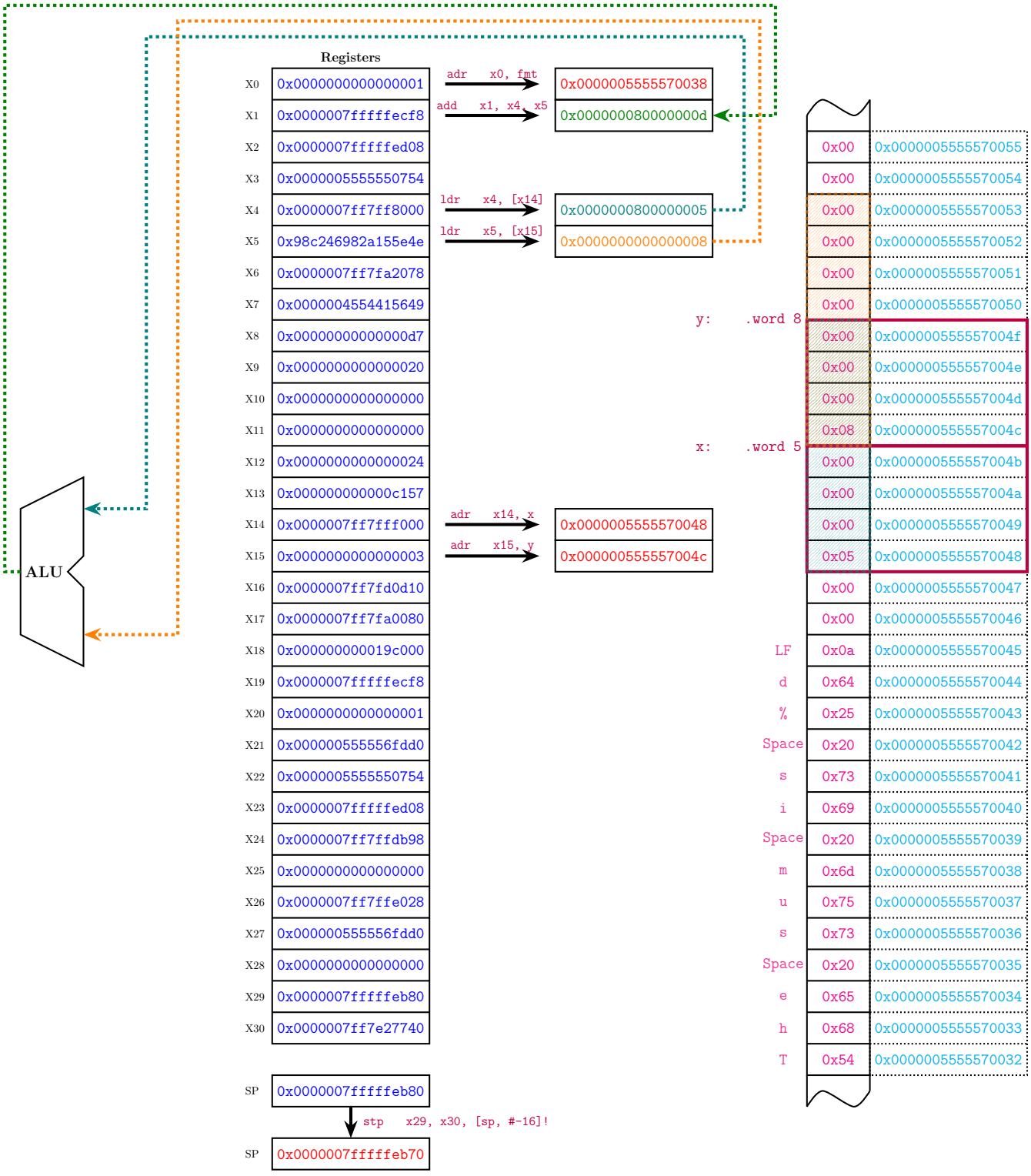
```c
#include <stdio.h>
static int x = 5;
static int y = 8;
int main(void) {
   int sum;
   sum = x + y;
   printf("The sum is %d\n",sum);
   return 0;
}
```

There are six basic arithmetic operations:

`add`	Addition
`adc`	Addition with Carry
`sub`	Subtract
`suc`	Subtraction with Carry
`neg`	Negate
`ngc`	Negate with Carry

```
        .data
fmt:    .asciz   "The sum is %d\n"
        .algin   2
x:      .word    5
y:      .word    8
        .text
        .type    main, %function
        .global  main
main:
        stp   x29, x30, [sp, #-16]!   // Push FP, LR onto the stack
        // sum = x + y
        adr   x14, x                  // Calculate address of x
        adr   x15, y                  // Calculate address of y
        ldr   x4, [x14]               // Load x
        ldr   x5, [x15]               // Load y
        add   x1, x4, x5              // x1 = x4 + x5

        //  printf("The sum is %d\n", sum)
        adr   x0, fmt              // Calculate address of fmt
        bl    printf               // Call the printf function

        // return 0
        mov   w0, #0
        ldp   x29, x30, [sp], #16  // Pop FP and LR from the stack
        ret                        // Return from main
        .size    main,(. - main)
```

- **Data Segment**:
  - `fmt` holds `"The sum is %d\n\0"`.
  - `x` contains the integer `5`.
  - `y` contains the integer `8`.
- **Stack Frame during main Execution**:
  - Temporary space for `x29` and `x30`.
  - Stack pointer adjusts as needed for the function call and return.


Registers
X0 0x0000000000000001
X1 0x0000007fffffecf8
X2 0x0000007fffffed08
X3 0x0000005555550754
X4 0x0000007ff7ff8000
X5 0x98c246982a155e4e
X6 0x0000007ff7fa2078
X7 0x0000004554415649
X8 0x00000000000000d7
X9 0x0000000000000020
X10 0x0000000000000000
X11 0x0000000000000000
X12 0x0000000000000024
X13 0x000000000000c157
X14 0x0000007ff7fff000
X15 0x0000000000000003
X16 0x0000007ff7fd0d10
X17 0x0000007ff7fa0080
X18 0x000000000019c000
X19 0x0000007fffffecf8
X20 0x0000000000000001
X21 0x000000555556fdd0
X22 0x0000005555550754
X23 0x0000007fffffed08
X24 0x0000007ff7ffdb98
X25 0x0000000000000000
X26 0x0000007ff7ffe028
X27 0x000000555556fdd0
X28 0x0000000000000000
X29 0x0000007fffffeb80
X30 0x0000007ff7e27740
SP 0x0000007fffffeb80
stp x29, x30, [sp, #-16]!
SP 0x0000007fffffeb70
adr x0, fmt
0x0000005555570038
add x1, x4, x5
0x000000080000000d
ldr x4, [x14]
0x0000000800000005
ldr x5, [x15]
0x0000000000000008
adr x14, x
0x0000005555570048
adr x15, y
0x000000555557004c
ALU
y: .word 8
x: .word 5
0x00 0x0000005555570055
0x00 0x0000005555570054
0x00 0x0000005555570053
0x00 0x0000005555570052
0x00 0x0000005555570051
0x00 0x0000005555570050
0x00 0x000000555557004f
0x00 0x000000555557004e
0x00 0x000000555557004d
0x08 0x000000555557004c
0x00 0x000000555557004b
0x00 0x000000555557004a
0x00 0x0000005555570049
0x05 0x0000005555570048
0x00 0x0000005555570047
0x00 0x0000005555570046
LF 0x0a 0x0000005555570045
d 0x64 0x0000005555570044
% 0x25 0x0000005555570043
Space 0x20 0x0000005555570042
s 0x73 0x0000005555570041
i 0x69 0x0000005555570040
Space 0x20 0x0000005555570039
m 0x6d 0x0000005555570038
u 0x75 0x0000005555570037
s 0x73 0x0000005555570036
Space 0x20 0x0000005555570035
e 0x65 0x0000005555570034
h 0x68 0x0000005555570033
T 0x54 0x0000005555570032

4.2 Shift Operations

4.3 Multiply Operations with Overflow

4.4 Multiply Operations with 64-bit results

4.5 Multiply Operations with 128-bit results

4.6 Division Operations

4.7 Comparison Operations

4.8 Conditional Operations

5 Structured Programming

6 Section Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex. Fusce quis mauris egestas lacus commodo venenatis. Ut at arcu lectus. Donec et urna nunc. Morbi eu nisl cursus sapien eleifend tincidunt quis quis est. Donec ut orci ex. Praesent ligula enim, ullamcorper non lorem a, ultrices volutpat dolor. Nullam at imperdiet urna. Pellentesque nec velit eget est pretium.[3]

[3] This is a sidenote. This template features a large margin specifically so you can put notes, figures, tables and other things into it as additional material to the main content in the text block.

Donec in elit ac ante vestibulum rhoncus. Pellentesque ligula tortor, aliquet malesuada nulla tristique vitae. Aliquam mi sem, varius eu pellentesque et, tristique nec quam. Vestibulum pellentesque in dui et venenatis. Sed malesuada elit pellentesque sapien aliquet porta. In at facilisis diam. Duis id ante tellus.[4]

[4] This sidenote has been pushed down the page manually with an optional parameter, otherwise it would be right under the one above.

6.1 Subsection Title

In diam libero, vulputate quis accumsan non, auctor in ipsum. Praesent cursus velit eget lacus sodales porta. Proin quis risus ut velit euismod scelerisque ut sed neque. Cras sagittis, dolor ac ullamcorper auctor, tortor dui facilisis diam, at sagittis nisi ipsum a neque. Nullam vel mattis nisi. Ut interdum ut diam at ornare. Nulla ultrices elit justo, vitae tristique massa vulputate sit amet.

Vestibulum erat felis, cursus vitae convallis ac, commodo eu nisi. Nulla facilisi. Mauris dignissim nisi felis, a mollis ex accumsan vel. Suspendisse bibendum vitae nibh in suscipit. Vestibulum et finibus eros. Nulla facilisi. Cras luctus aliquam finibus. In nec justo nec orci malesuada faucibus.

This sidenote isn't numbered in the text or margin. This is useful for notes that apply anywhere on the page instead of one particular place.

6.1.1 Subsubsection Title

This is an example of a full width paragraph... Curabitur id placerat orci. Vivamus pulvinar augue ac feugiat blandit. Donec in ultricies mi. Nam eu lacus ac augue aliquet consectetur. Praesent dui risus, sollicitudin nec felis ut, posuere ultricies dolor. Sed massa nulla, dignissim eget sem sit amet, eleifend fermentum dui. Phasellus consequat sem vel turpis finibus, a aliquam risus malesuada.

Maecenas consectetur metus at tellus finibus condimentum. Proin arcu lectus, ultrices non tincidunt et, tincidunt ut quam.

Integer luctus posuere est, non maximus ante dignissim quis. Nunc a cursus erat. Curabitur suscipit nibh in tincidunt sagittis. Nam malesuada vestibulum quam id gravida. Proin ut dapibus velit. Vestibulum eget quam quis ipsum semper convallis. Duis consectetur nibh ac diam dignissim, id condimentum enim dictum. Nam aliquet ligula eu magna pellentesque, nec sagittis leo lobortis. Aenean tincidunt dignissim egestas. Morbi efficitur risus ante, id tincidunt odio pulvinar vitae.

Paragraph Title Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex.

The section titles below show how multi-line section titles look at the 3 top levels.

7 Fusce eleifend porttitor arcu, id accumsan elit pharetra eget

Section, subsection and subsubsection titles can span multiple lines, as shown here. Make sure to put a shorter version of these long titles in the optional parameter to the section commands so the title output to the table of contents is the short version.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

7.1 Phasellus sit amet enim efficitur, aliquam nulla id, lacinia mauris viverra libero ac magna

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

7.1.1 In mi mauris, finibus non faucibus non, imperdiet nec leo. In erat arcu, tincidunt nec aliquam et, volutpat eget

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

8 Font Examples

8.1 Font Sizes

The default font size for the document is 12pt, represented by \normalsize. The standard LaTeX font size commands modify this to be smaller or larger as needed.

\tiny \scriptsize \footnotesize \small \normalsize

\large \Large \LARGE \huge \Huge

8.2 Font Families

IBM Plex Sans Text

IBM Plex Serif Text

`IBM Plex Mono Text`

The sans family is the default, as is standard in the business world. Use the serif family to accentuate text, such as for quotations. The mono family is best used where it's important that all characters are the same width, such as for numbers in a table or for code.

8.3 Font Weights

ExtraLight Light Normal **SemiBold Bold**

8.4 Condensed Fonts

Plex Sans Normal

Plex Sans Condensed

Condensed fonts can be useful if horizontal space is at a premium. You might want to use the condensed font in a wide table.

9 Quotations

Proin mollis urna posuere fringilla. Curabitur finibus, neque vitae vestibulum vestibulum, dolor sapien tincidunt augue, vel porta mauris metus nec mauris. Integer erat magna, porta id erat sed, lacinia volutpat erat. Nulla fermentum tellus arcu, eu iaculis ipsum malesuada aliquam. Duis et lacus maximus, consectetur metus et, eleifend arcu. Vestibulum condimentum diam vitae diam tincidunt viverra.

> *"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor. Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in."*
>
> *— John Smith, 1972*

"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie."

— John Smith, 1972

Suspendisse tempus odio sit amet volutpat suscipit. Pellentesque ornare libero lacus, non fringilla dolor placerat in. Ut maximus ullamcorper lectus, a pharetra mi sagittis aliquet. Scelerisque augue sed mi fringilla, vel dapibus ligula finibus. Sed ornare velit sem, ac venenatis velit dignissim. Vestibulum ultrices mi at tincidunt condimentum.

10 Table Examples

This statement automatically references the table below using its label: Table 8.

Table 8: Text block table caption.

Prospect	Industry	Revenue
Gerlach Inc	Business Development	$3M
Doyle and Sons	Law	$1M
Heathcote Group	Consulting	$12M
Goyette Inc	Advertising	$5M
Holzdeppe GmbH	Manufacturing	$23M
Bienias AG	Accounting	$2.5M

Table 7: Margin table caption.

Year	Qtr.	Perf.
20XX	Q1	0.5%
20XX	Q2	26.5%
20XX	Q1	35.4%
20XX	Q4	41.3%

Table 9: Full width table caption.

#	Prospect	Industry	Revenue	Employees
1	Gerlach Inc	Business Development	$3M	65
2	Doyle and Sons	Law	$1M	15
3	Heathcote Group	Consulting	$12M	250
4	Goyette Inc	Advertising	$5M	100
5	Holzdeppe GmbH	Manufacturing	$23M	75
6	Bienias AG	Accounting	$2.5M	40

11 Figure Examples

This statement automatically references the figure below using its label: Figure 9.

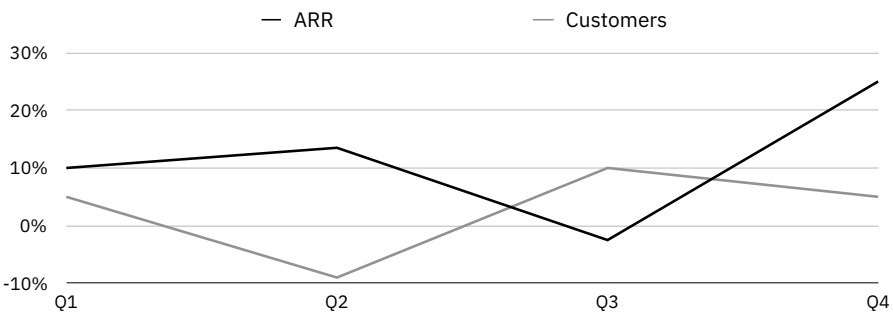


Figure 9: Text block figure caption.

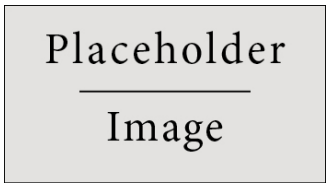


Figure 8: Margin figure caption.

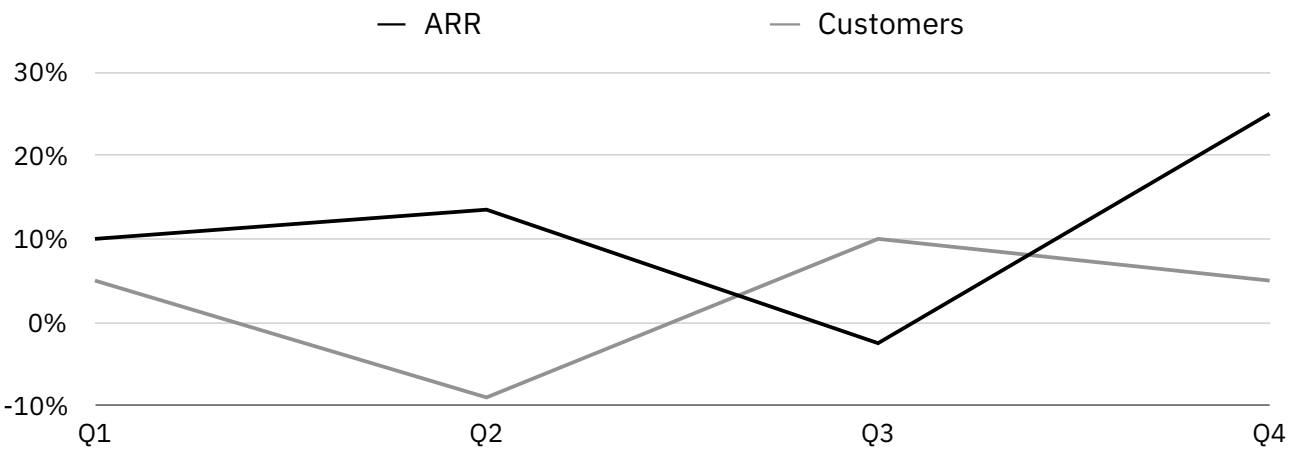


Figure 10: Full width figure caption.

12 List Examples

12.1 Bullet Point List

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor.

- First bullet point item
  - First indented bullet point item
  - Second indented bullet point item
    - First second-level indented bullet point item
    - Second second-level indented bullet point item
  - Third indented bullet point item
- Second bullet point item
- Third bullet point item

Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in. Nunc fermentum nisi imperdiet lectus tincidunt vestibulum at ac elit. Nulla mattis nisl eu malesuada suscipit.

Bullet point lists can also be created in the margin. For these, we can remove the usual left margin to increase the available horizontal space:

- Bullet item one.
- Bullet item two.
- Bullet item three.

12.2 Numbered List

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor.

Numbered lists can also be created in the margin. For these, we can remove the usual left margin to increase the available horizontal space:

1. Numbered item one.
2. Numbered item two.
3. Numbered item three.

1. First numbered item
    a. First indented numbered item
    b. Second indented numbered item
        i. First second-level indented numbered item
        ii. Second second-level indented numbered item
    c. Third indented numbered item
2. Second numbered item
3. Third numbered item

Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in. Nunc fermentum nisi imperdiet lectus tincidunt vestibulum at ac elit. Nulla mattis nisl eu malesuada suscipit.

12.3 Description List

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie. Sed dictum nisl non aliquet porttitor.

Description lists can also be created in the margin:

A1 Description item one.
B1 Description item two.
C1 Description item three.

Item One Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent porttitor arcu luctus, imperdiet urna iaculis, mattis eros. Pellentesque iaculis odio vel nisl ullamcorper, nec faucibus ipsum molestie.

Item Two Sed dictum nisl non aliquet porttitor.

Subitem Maecenas consectetur metus at tellus finibus condimentum. Proin arcu lectus, ultrices non tincidunt et, tincidunt ut quam. Integer luctus posuere est, non maximus ante dignissim quis.

Subsubitem Maecenas consectetur metus at tellus finibus condimentum. Proin arcu lectus, ultrices non tincidunt et, tincidunt ut quam. Integer luctus posuere est, non maximus ante dignissim quis.

Item Three Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in. Nunc fermentum nisi imperdiet lectus tincidunt vestibulum at ac elit. Nulla mattis nisl eu malesuada suscipit.

Etiam vulputate arcu dignissim, finibus sem et, viverra nisl. Aenean luctus congue massa, ut laoreet metus ornare in.

13 Referencing Citations

This statement requires citation [**Smith:2024jd**].

This statement requires multiple citations [**Smith:2024jd**, **Smith:2023qr**].

This short citation is in the margin[5].

[5] **Smith:2023qr**

This long citation is in the margin[6].

[6] **Smith:2024jd**

This statement has an in-text citation: **Smith:2024jd**.

14 Link Examples

This is a URL link: DuckDuckGo.

This is a email link: example@example.com.

This is a monospaced URL link: `https://duckduckgo.com`.

Links can be clicked in the PDF to navigate to the linked website or email address.

15 Equation

$$\cos^3\theta = \frac{1}{4}\cos\theta + \frac{3}{4}\cos 3\theta \tag{1}$$

This statement automatically references the equation above using its label: Equation 1.

16 International Support

àáâäãåèéêëìíîïòóôöõøùúûüÿýñçčšž

ÀÁÂÄÃÅÈÉÊËÌÍÎÏÒÓÔÖÕØÙÚÛÜŸÝÑ

ßÇŒÆČŠŽ

Plex is a very high quality typeface produced by IBM. It includes extensive international support and characters.

17 Displaying Code

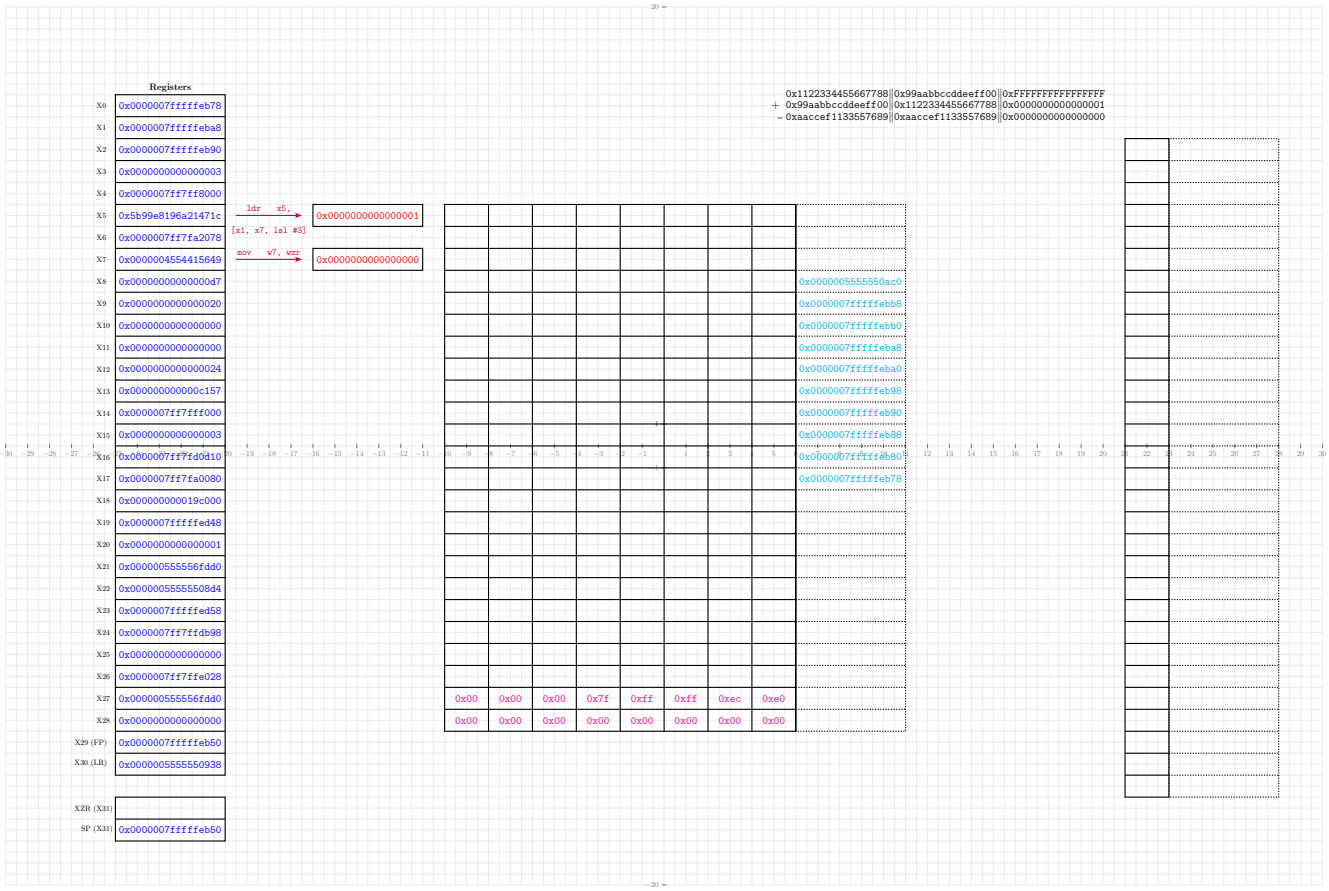
The block below is a code listing. It displays code in an easy to use way with line numbers for quick reference to specific parts of the code.

```
{
   "city": [
      {
         "id": 1,
         "name": "Toronto",
         "country": "Canada",
         "population": 6200000
      },
      {
         "id": 2,
         "name": "New York",
         "country": "United States of America",
         "population": 8800000
      }
   ]
}
```

Appendices

Big Number Addition

```
.global bn_add_asm
.type bn_add_asm, %function

/* void bn_add_asm(
        uint64_t* res,
        const uint64_t* op1,
        const uint64_t* op2,
        int32_t n);
    x0: res (Address)
    x1: op1 (Address)
    x2: op2 (Address)
    x3: n
    x4: *res (Contents)
    x5: *op1 (Contents)
    x6: *op2 (Contents)
    x7: counter */
bn_add_asm:
        mov     w7, wzr

        ldr     x5, [x1, x7, lsl #3]
        ldr     x6, [x2, x7, lsl #3]
        adds    x4, x5, x6
        str     x4, [x0, x7, lsl #3]
        add     w7, w7, #1

LOOP:
        ldr     x5, [x1, x7, lsl #3]
        ldr     x6, [x2, x7, lsl #3]
        adcs    x4, x5, x6
        str     x4, [x0, x7, lsl #3]
        add     w7, w7, #1
        cmp     w3, w7
        bgt     LOOP

END_LOOP:
        adc     w0, wzr, wzr

        ret
        .size bn_add_asm, (. - bn_add_asm)
```



A Big Nu

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex. Fusce quis mauris egestas lacus commodo venenatis. Ut at arcu lectus. Donec et urna nunc. Morbi eu nisl cursus sapien eleifend tincidunt quis quis est. Donec ut orci ex. Praesent ligula enim, ullamcorper non lorem a, ultrices volutpat dolor. Nullam at imperdiet urna. Pellentesque nec velit eget euismod pretium.

B Appendix Section

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex. Fusce quis

mauris egestas lacus commodo venenatis. Ut at arcu lectus. Donec et urna nunc. Morbi eu nisl cursus sapien eleifend tincidunt quis quis est. Donec ut orci ex. Praesent ligula enim, ullamcorper non lorem a, ultrices volutpat dolor. Nullam at imperdiet urna. Pellentesque nec velit eget euismod pretium.

C Appendix Section

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam auctor mi risus, quis tempor libero hendrerit at. Duis hendrerit placerat quam et semper. Nam ultricies metus vehicula arcu viverra, vel ullamcorper justo elementum. Pellentesque vel mi ac lectus cursus posuere et nec ex. Fusce quis mauris egestas lacus commodo venenatis. Ut at arcu lectus. Donec et urna nunc. Morbi eu nisl cursus sapien eleifend tincidunt quis quis est. Donec ut orci ex. Praesent ligula enim, ullamcorper non lorem a, ultrices volutpat dolor. Nullam at imperdiet urna. Pellentesque nec velit eget euismod pretium.