EECS 665: Lab 7 Report

For this lab, I will be looking into the MIPS and ARM processor architectures.

According to https://scottiestech.info/2011/11/24/what-is-an-arm-processor-and-whats-the-big-deal/, the ARM architecture is a RISC architecture. It sacrifices a high number of instructions needed for a program in order to maximize the performance of an individual instruction. The ARM architecture's cache is layed out in the following way. Each cache is "physically indexed and physically addressed. The cache sizes are configurable with sizes in the range of 1 to 64KB, but the maximum clock frequency might be affected if you increase the cache sizes beyond 16KB. Both the instruction cache and the data cache are capable of providing two words per cycle for all requesting sources.
The cache way size can be varied between 1KB and 16KB in powers of 2. A 1KB cache size must be implemented as a 1 way cache, and a 2KB cache must be implemented as a 2 way cache. All other cache sizes must be implemented as 4 way set associative. The cache line length is fixed at eight words (32 bytes)." (http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0338g/Chdjdeci.html)
The ARM architecture has a fixed width of 32 bits per instruction (https://www.coursera.org/learn/comparch/lecture/WXq25/isa-characteristics). The data types supported and their bit-widths can be found in the following table taken from http://www.keil.com/support/man/docs/armcc/armcc_chr1359125009502.htm .

Size and alignment of basic data types

The following table gives the size and natural alignment of the basic data types.

Table 10-2 Size and alignment of data types

Type	Size in bits	Natural alignment in bytes	Range of values
char	8	1 (byte-aligned)	0 to 255 (unsigned) by default. –128 to 127 (signed) when compiled with `--signed_chars`.
signed char	8	1 (byte-aligned)	–128 to 127
unsigned char	8	1 (byte-aligned)	0 to 255
(signed) short	16	2 (halfword-aligned)	–32,768 to 32,767
unsigned short	16	2 (halfword-aligned)	0 to 65,535
(signed) int	32	4 (word-aligned)	–2,147,483,648 to 2,147,483,647
unsigned int	32	4 (word-aligned)	0 to 4,294,967,295
(signed) long	32	4 (word-aligned)	–2,147,483,648 to 2,147,483,647
unsigned long	32	4 (word-aligned)	0 to 4,294,967,295
(signed) long long	64	8 (doubleword-aligned)	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	64	8 (doubleword-aligned)	0 to 18,446,744,073,709,551,615
float	32	4 (word-aligned)	1.175494351e-38 to 3.40282347e+38 (normalized values)
double	64	8 (doubleword-aligned)	2.22507385850720138e-308 to 1.79769313486231571e+308 (normalized values)
long double	64	8 (doubleword-aligned)	2.22507385850720138e-308 to 1.79769313486231571e+308 (normalized values)
wchar_t	16 32	2 (halfword-aligned) 4 (word-aligned)	0 to 65,535 by default. 0 to 4,294,967,295 when compiled with `--wchar32`.
All pointers	32	4 (word-aligned)	Not applicable.
bool (C++ only)	8	1 (byte-aligned)	false or true
_Bool (C only[a])	8	1 (byte-aligned)	false or true

The data types can be stored in either big endian or little endian depending on how the implementer of the ISA chooses according to
http://www.keil.com/support/man/docs/armcc/armcc_chr1359125009502.htm .
Information about the ARM registers can be found in the following table taken from
http://www.davespace.co.uk/arm/introduction-to-arm/registers.html

Registers

ARM has sixteen registers visible at any one time. They are named R0 to R15. All are 32 bits wide.

R0	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13	R14	R15

The registers may also be referred to by the following aliases:

A1	A2	A3	A4	V1	V2	V3	V4 WR	V5	V6 SB	V7 SL	V8 FP	IP	SP	LR	PC

All of the registers are general purpose, save for:

- R13 / SP
  - which holds the *stack pointer*.
- R14 / LR
  - the link register which holds the callers's return address.
- R15 / PC
  - which holds the *program counter*.

In addition to the main registers there is also a status register:

CPSR

CPSR is the *current program status register*. This holds flags: results of arithmetic and logical operations.

The instruction set is three address. Each instruction needs operand1, operand 2, and the destination of the operation. https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/

The MIPS processor architecture is also a RISC architecture.
(http://booksite.elsevier.com/samplechapters/9780120884216/Sample_Chapters/02~Chapter_1.pdf)

The MIPS instructions are all fixed to 32 bits wide
(http://booksite.elsevier.com/samplechapters/9780120884216/Sample_Chapters/02~Chapter_1.pdf)

The Data types supported by the MIPS ISA are bytes(8 bits), halfwords(2 bytes), and words(4 bytes) a char requires 1 byte while an integer requires 4 bytes. (http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm)

MIPS is a big endian architecture http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm

The MIPS registers are all 32 bits wide and are organized in the following way (
http://logos.cs.uic.edu/366/notes/mips%20quick%20tutorial.htm)

Register Number	Alternative Name	Description
0	zero	the value 0
1	$at	(**a**ssembler **t**emporary) reserved by the assembler
2-3	$v0 - $v1	(**v**alues) from expression evaluation and function results
4-7	$a0 - $a3	(**a**rguments) First four parameters for subroutine. Not preserved across procedure calls
8-15	$t0 - $t7	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. Not preserved across procedure calls
16-23	$s0 - $s7	(**s**aved values) - Callee saved. A subroutine using one of these must save original and restore it before exiting. Preserved across procedure calls
24-25	$t8 - $t9	(**t**emporaries) Caller saved if needed. Subroutines can use w/out saving. These are in addition to $t0 - $t7 above. Not preserved across procedure calls.
26-27	$k0 - $k1	reserved for use by the interrupt/trap handler
28	$gp	**g**lobal **p**ointer. Points to the middle of the 64K block of memory in the static data segment.
29	$sp	**s**tack **p**ointer Points to last location on the stack.
30	$s8/$fp	**s**aved value / **f**rame **p**ointer Preserved across procedure calls
31	$ra	**r**eturn **a**ddress

MIPS is a 3 address ISA. Each instruction requires operand1, operand2, and destination for the instruction.