What you will learn.
In this chapter you will learn:

1. two of the three formats for MIPS instructions.
2. what opcodes and functions are for MIPS operations
3. how to format Immediate (I) and Register (R) instructions in machine code.
4. how to represent MIPS instructions in hexadecimal.
5. how to use MARS to check your translations from assembly language to machine code.

Chapter 4 Translating Assembly Language into Machine Code

Everything in a computer is a binary value. Numbers are binary, characters are binary, and even the instructions to run a program are binary. Therefore, the assembly language that has been presented cannot be what the computer understands. This assembly code must be converted to binary values. These binary values are called machine code. This chapter will explain how to convert the assembly instructions that have been covered so far into machine code.

Chapter 4. 1 Instruction formats

The MIPS computer is organized in a 3-address format. Generally, all instructions will have 3 addresses associated with them. For example, the instruction "`add` R_d`,` R_s`,` R_t" uses three registers, the first address is the destination of the result and the second and third are the two inputs to the ALU. The instruction "`addi` R_t`,` R_s`, immediate`" also uses three addresses, but in this case the third address is an immediate value.

In MIPS there are only 3 ways to format instructions. They are the R-format (register), the I-format (immediate), and the J-format (jump). This chapter will only cover R-format and I-format instructions. The R-format and I-format are shown below.

opcode	rs	rt	rd	Shift (shamt)	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Figure 4-1: R format instruction

opcode	rs	rt	Immediate Value
6 bits	5 bits	5 bits	16 bits

Figure 4-2: I format instruction

The R format instruction is used when the input values to the ALU come from two registers. The I format instruction is used when the input values to the ALU come from one register and one immediate value.
The fields in these instructions are as follows:

- Opcode - a 6-bit code which specifies the operation. These codes can be found for each instruction on a MIPS Green Sheet or Reference Sheet.

  Opcodes are stored as a 2-digit number. The first number is 2 bits wide and has values from 0-3. The second number is 4 bits wide and has values from 0x0 to 0xf. So, for example the `lw` instruction has an opcode of 23, or $10\ 0011_2$. The `addi` has an opcode of 8, or 00 1000.

  All R-format operators have an opcode of 0 and are specified as an opcode/function. So, the `add` opcode is specified by an opcode/function of 0/20. The operation to be performed by the ALU is contained in the function, which is the last field in the R instruction. Functions, like opcodes, are specified as a 2-bit number and a 4-bit number.

- `rd` – always the destination register if specified
- `rs` – the register of the first operand/input supplied to the ALU
- `rt` – when there is a `rd` in the instruction, `rt` is the register of the second operand/input supplied to the ALU. If there is no `rd`, `rt` is the destination or target register
- Note that `rd`, `rs`, and `rt` are 5 bits wide, which allows the addressing of the 32 general purpose registers.
- `shamt` - The number of bits to shift if the operation is a shift operation, otherwise it is 0. It is 5 bits wide, which allows for shifting of all 32 bits in the register.
- `funct` - For an R type instruction, this specifies the operation for the ALU.
- `Immediate Value` - The immediate value for instructions which use them.

In MARS the machine code which is generated from the assembly code is shown in the Code column, as in Figure 4-3. A good way to check if you have translated your code correctly is to see if your translation matches the code in this column.

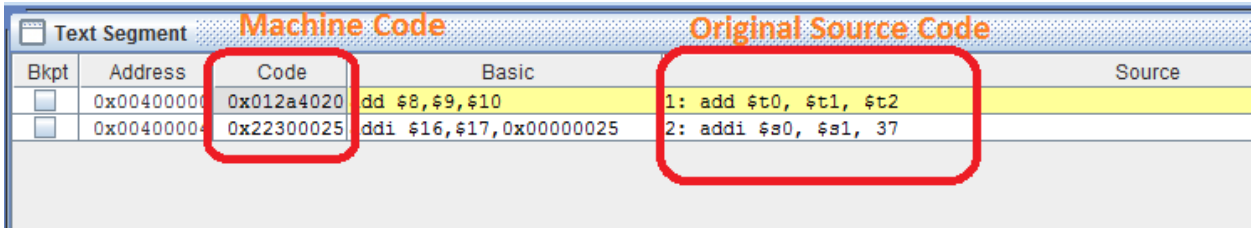

Figure 4-3: Machine Code example

The rest of the sections of this chapter are intended to help makes sense of this machine code.

Chapter 4. 2 Machine code for the `add` instruction

This section will translate the following `add` instruction to machine code.

```
add $t0, $t1, $t2
```

The MIPS Green Sheet specifies the `add` instruction as an R-format instruction and the opcode/function for the `add` as 0/20. The opcode/function field is made up of two numbers, the first is the opcode, and the second is the function. Note that the function is used only for R-format instructions. If the instruction has a function, the number to the left of the "/" is the opcode, and the number to the right of the "/" is the function. If there is only one number with no "/", it is the opcode, and there is no function.

Both the opcode and the function are 6 bits, divided into a 2-bit number and a 4-bit number. So the first number in the opcode and function is 0…3, and the second is 0…f. Both are generally called hex values, so this text will do so as well. So, the 6 bits for the opcode translate to 00 0000, and the 6 bits for the function translate to 10 0000. These are placed into the opcode and function fields of the R-format instruction shown in Figure 4-4 below.

Register R_d is `$t0`. `$t0` is also register `$8`, or 01000, so 0 1000 is placed in the R_d field.

Register R_s is `$t1`. `$t1` is also register `$9`, or 01001, so 0 1001 is placed in the R_s field

Register R_t is `$t2`. `$t2` is also register `$10`, or 01010, so 0 1010 is placed in the R_t field.

The `shamt` is 0 0000 as there are no bits being shifted.

The result is the following R-format instruction.

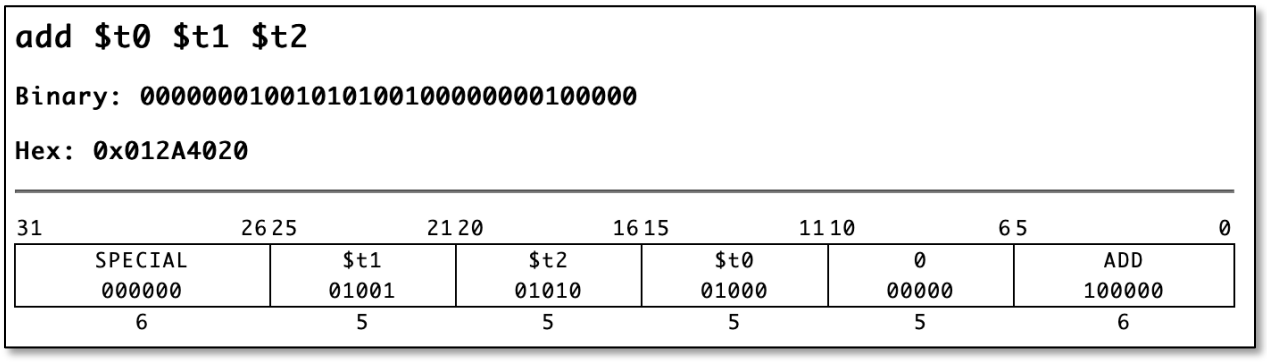

Figure 4-4: Machine code for add $t0, $t1, $t2

Notice that the order of the registers and the opcode in machine code (Figure 4-4) is different than in assembly language (Figure 4-1). Assembly language (`add $t0, $t1, $t2`) is designed to be easy for humans to read whereas machine code is dictated by the computer hardware/circuitry.

The instruction "`add $t0, $t1, $t2`" translate into the bit string:

00000001001010100100000000100000

This is as hard to type as it is to read. To make it readable, the bits are divided into groups of 4 and these 4-bit values translated into hex. This results in the following instruction:

	0000	0001	0010	1010	0100	0000	0010	0000
0x	0	1	2	a	4	0	2	0

This can be checked by entering the instruction in MARS and assembling it to see the resulting machine code. Be careful with the order of the registers when translating into machine code, as the R_d is the last register in machine code.

Chapter 4. 3 Machine code for the `sub` instruction

This section will translate the following `sub` instruction to machine code.

```
sub $s0, $s1, $s2
```

The MIPS Green Sheet specifies the `sub` instruction as an R-format instruction and the opcode/function for the `sub` as 0/22. This means the 6 bits for the opcode are 00 0000 and the 6 bits for the function are 10 0010.

Register R_d is `$s0` is also register `$16`, or 1 0000.

Register R_s is `$s1` is also register `$17`, or 1 0001.

Register R_t is `$s2` is also register `$18`, or 1 0010.

The `shamt` is 0 0000 as there are no bits being shifted.

The result is the following R-format instruction.

sub $s0 $s1 $s2

Binary: 00000010001100101000000000100010

Hex: 0x02328022

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL	$s1	$s2	$s0	0	SUB
000000	10001	10010	10000	00000	100010
6	5	5	5	5	6

Figure 4-5: Machine code for sub $s0, $s1, $s2

To write this instruction's machine code, the bits are organized in groups of 4, and hex values given. This results in the number 0000 0010 0011 0010 1000 0000 0010 0010_2 or 0x02328022.

Chapter 4. 4 Machine code for the `addi` instruction

This section will translate the following `addi` instruction to machine code.

```
addi $s2, $t8, 37
```

The MIPS Green Sheet specifies the `addi` instruction as an I-format instruction and the opcode/function for the `addi` as 8 (note that there is no function for an I-format instruction). This means the 6 bits for the opcode are 001000.

Register R_t is `$s2` is also register `$18`, or 1 0010.

Register R_s is `$t8` is also register `$24`, or 1 1000.

The immediate value is 37, or 0x0025.

The Result is the following I-format instruction.

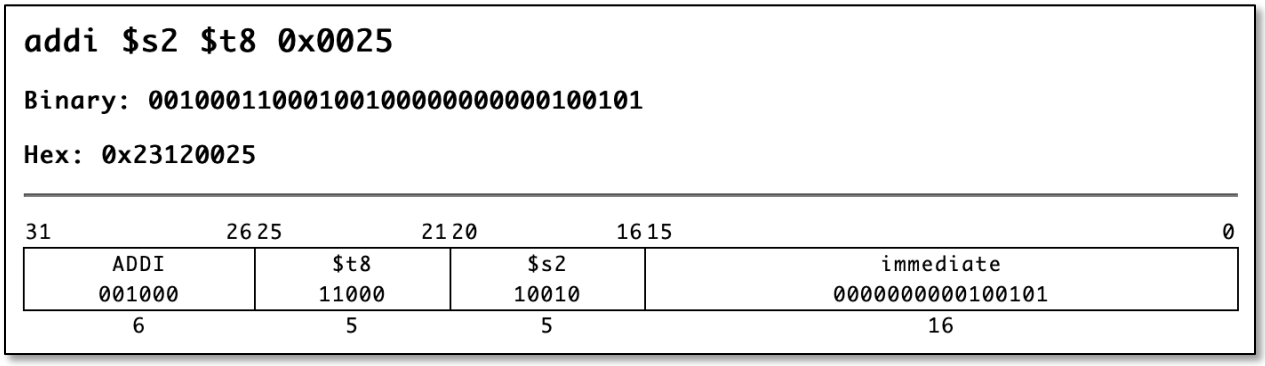

Figure 4-6: Machine code for addi $s2, $t8, 37

To write this instruction's machine code, the bits are organized in groups of 4, and hex values given. This results in the number 0010 0011 0001 0010 0000 0000 0010 0101_2 or 0x23120025.

When translating the I-format instruction, be careful to remember that R_t is destination register.

Chapter 4. 5 Machine code for the `sll` instruction

This section will translate the following SLL instruction to machine code.

```
sll $t0, $t1, 10
```

The MIPS Green Sheet specifies the `sll` instruction as an R-format instruction and the opcode/function for the `sll` as 0/00. This means the 6 bits for the op code are 00 0000 and the 6 bits for the function are 00 0000.

Register R_d is `$t0` is also register `$8`, or 0 1000.

Register R_s is not used.

Register R_t is `$t1` is also register `$9`, or 0 1001.

The `shamt` is 10, or 0xa.

The Result is the following R-format instruction.

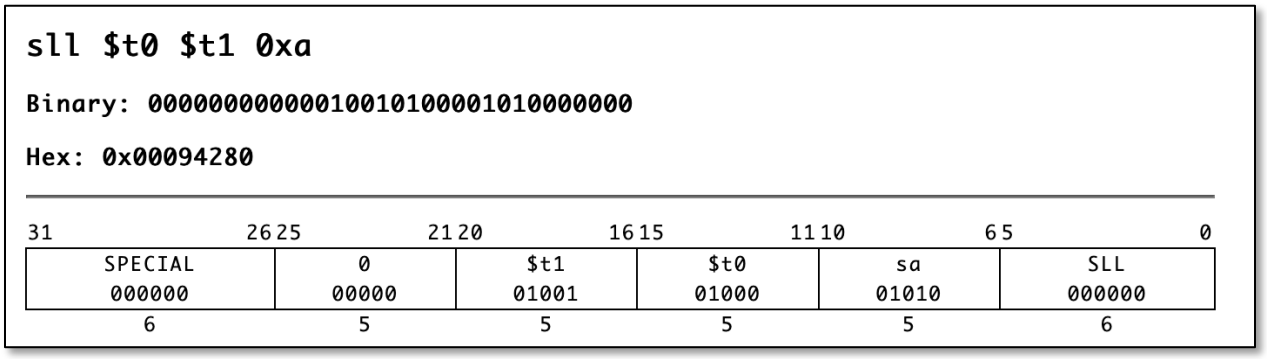

Figure 4-7: Machine code for sll $t0, $t1, 10

To write this instruction's machine code, the bits are organized in groups of 4, and hex values given. This results in the number 0000 0000 0000 1001 0100 0010 1000 0000_2 or 0x00094280.

Chapter 4. 6 Exercises

1) Translate the following MIPS assembly language program into machine code.

```
.text
.globl main
main:
    ori $t0, $zero, 15
    ori $t1, $zero, 3
    add $t1, $zero $t1
    sub $t2, $t0, $t1
    sra $t2, $t2, 2
    mult $t0, $t1
    mflo $a0
    ori $v0, $zero, 1
    syscall
    addi $v0, $zero, 10
    syscall

.data
    result: .asciiz "15 * 3 is "
```

2) Translate the following MIPS machine code into MIPS assembly language.

```
0x2010000a
0x34110005
0x012ac022
0x00184082
0x030f9024
```