# 605.611 - Foundations of Computer Architecture
# Assignment 06 - Branching and Jumps

Sabbir Ahmed

March 9, 2021

1. Translate statements 1, 2, and 5 into machine code. Assumes the program loads at 0x0040 0000.

```
1 label_1:
2     beq $zero, $zero, label_2
3     j label_3
4     addi $t0, $t1, 100
5     sub $t0, $t1, $t2
6 label_2:
7     beq $zero, $zero, label_1
8 label_3:
9     addi $t0, $t1, 100
```

**Answer:** The statements 1, 2, and 5 are

```
beq $zero, $zero, label_2
j label_3
beq $zero, $zero, label_1
```

Putting the program with its address space:

| Address | Instruction |
|---|---|
| 0x0040 0000 | label_1:<br>beq $zero, $zero, label_2 |
| 0x0040 0004 | j label_3 |
| 0x0040 0008 | addi $t0, $t1, 100 |
| 0x0040 000C | sub $t0, $t1, $t2 |
| 0x0040 0010 | label_2:<br>beq $zero, $zero, label_1 |
| 0x0040 0014 | label_3:<br>addi $t0, $t1, 100 |

The addresses of the labels are as follows:

| Address | Instruction |
|---|---|
| 0x0040 0000 | label_1 |
| 0x0040 0010 | label_2 |
| 0x0040 0014 | label_3 |

Replacing addresses in absolute jumps:

| Address | Instruction |
|---|---|
| 0x0040 0000 | beq $zero, $zero, label_2 |
| 0x0040 0004 | j 0x0040 0014 |
| 0x0040 0010 | beq $zero, $zero, label_1 |

For the relative branches, label_2 is +3 PC (0x0000 0003) from label_1 and label_1 is -5 PC (0xFFFF FFFB) from label_2. Therefore,

| Address | Instruction |
|---|---|
| 0x0040 0000 | beq $zero, $zero, 0x0000 0003 |
| 0x0040 0004 | j 0x0040 0014 |
| 0x0040 0010 | beq $zero, $zero, 0xFFFF FFFB |

(a) `beq $zero, $zero, 0x0000 0003`

$$
\begin{aligned}
\text{opcode(beq)} &= 04_{16} \\
&= 000100_2 \\
\text{rs(\$zero)} &= 00000_2 \\
\text{rt(\$zero)} &= 00000_2 \\
\text{sign\_ext\_imm(3)} &\Rightarrow 0003_{16} \\
&\Rightarrow 000100\ 00000\ 00000_2 \mid 0003_{16} \\
&\Rightarrow 0001\ 0000\ 0000\ 0000 \mid 0003_{16} \\
&\Rightarrow 10000003_{16}
\end{aligned}
$$

(b) `j 0x0040 0014`

$$
\begin{aligned}
\text{opcode(j)} &= 02_{16} \\
&= 000010_2 \\
\text{imm(0x0040 \ 0014)} &\Rightarrow 0040\ 0014_{16} \\
&\Rightarrow 000010\ 0000_2 \mid 40\ 0014_{16} \mid 00_2 \\
&\Rightarrow 000010\ 0000\ 0100\ 0000\ 0000\ 0000\ 0001\ 0100\ 00 \\
&\Rightarrow 0000\ 1000\ 0001\ 0000\ 0000\ 0000\ 0000\ 0101\ 0000 \\
&\Rightarrow 08100005_{16}
\end{aligned}
$$

(c) `beq $zero, $zero, 0xFFFF  FFFB`

$$
\begin{aligned}
\text{opcode(beq)} &= 04_{16} \\
&= 000100_2 \\
\text{rs(\$zero)} &= 00000_2 \\
\text{rt(\$zero)} &= 00000_2 \\
\text{sign\_ext\_imm(-5)} &\Rightarrow FFFB_{16} \\
&\Rightarrow 000100\ 00000\ 00000_2 \mid FFFB_{16} \\
&\Rightarrow 0001\ 0000\ 0000\ 0000 \mid FFFB_{16} \\
&\Rightarrow 1000FFFB_{16}
\end{aligned}
$$

6. Explain, in your own words, how a 26 bit address in the `jmp` instruction can access any executable statement in a program. This must be your own words. Copying something from the internet does not count.

   **Answer:** All of the instructions in a MIPS machine allocate 6 bits for their opcodes. This limits 26 bits for the address for the `jmp` instruction. However, the memory is configured in a MIPS machine such that some areas are protected and reserved from the program text, i.e. the kernel space, operating system instructions, etc. The program text is limited to 0x1000 0000, which leaves a highest address of 0x0fff ffff. The most significant bit of the addresses are therefore always 0 (in hexadecimal), and 4-bits are regained. The other 2 bits come for the 2 zeros on the least significant bits that account for the word alignment of the addresses.