CSI 333 – Lecture 17
MIPS Assembly Language (MAL): Part V
(MIPS Instruction Formats)
Each MIPS machine instruction occupies one word (32 bits).

Three instruction formats (called R-format, I-format and J-format).

Each instruction uses one of these three formats.

32 general registers: 5 bits needed to specify a register.
R-Format (or Register Format)

Note: Op and Funct together determine the operation.

**Example:** The instruction

```
add $8, $17, $18
```

is stored as follows.

```
000000 10001 10010 01000 00000 100000
$17 $18 $8
```

Above instruction in hex: 0x02324020

**Note:** The Op part of an instruction determines how the rest of the instruction is decoded.
I-Format (for Immediate Operands)

Note: The 16-bit immediate operand field is treated as a signed integer.

Example: The instruction

\[ \text{addi } \$5, \$4, 49 \]

is stored as follows.

\[
\begin{array}{cccc}
001000 & 00100 & 00101 & 0000000000110001 \\
\end{array}
\]

Exercise: Show the above instruction in hex.
I-Format (continued)

Note:  I-format is also used for Load/Store instructions and conditional branches.

Example:  The instruction

\[ \text{lw} \; \$8, \; 12(\$6) \]

is stored as follows.

\[
\begin{array}{cccc}
100011 & 00110 & 01000 & 0000000000001100 \\
\hline
\end{array}
\]

Notes:
- The format for \textit{sw} is similar (except that opcode = 101011).
- If the offset (displacement) is negative, it will be stored in 2’s complement form (to be discussed later).
- Examples of conditional jumps – later in this lecture.
J-Format (for j and jal Instructions)

![Diagram of J-Format]

Example: The instruction

\texttt{j iloop}

where the address of iloop is 19400 (hex: 4BC8) is stored as follows.

<table>
<thead>
<tr>
<th>Op</th>
<th>Target address</th>
</tr>
</thead>
<tbody>
<tr>
<td>000010</td>
<td>000000000001001011100100</td>
</tr>
</tbody>
</table>

19,400 (decimal)

Note: With 26 bits for target address, the range is about 67 million words.
Other Jump Instructions

**Instruction jr:**
- Can be used for very long jumps.
- Compiler can determine whether to use j or jr.
- Allows the target address to be 32 bits long (a value that can be stored in a register).
- Uses the R-format.

**Example:** The instruction

```
jr $17
```

is stored as follows.

```
| 000000 | 10001 | 00000 | 00000 | 00000 | 001000 |
```

$17

---

Not used

jr opcode
Other Jump Instructions (continued)

**Instruction** jalr:

- Purpose similar to jr.
- Allows the return address to be stored in a register (other than $31).
- Also uses the R-format.

**Example:** The instruction

\[
\text{jalr} \quad \$3, \$5
\]

is stored as follows.

```
  000000  0011  00000  00101  00000  001001
```

```plaintext
$3    $5
```

Not used Not used

jalr opcode
Conditional Jump Instructions

- Use the I-format. (Offset for the jump is specified as the immediate operand.)
- The offset is specified using \textit{PC-relative} mode of addressing.

\textbf{Example:} Consider the TAL instruction

\begin{verbatim}
bgtz  $9, end_loop
\end{verbatim}

\begin{verbatim}
000111 01001 00000
\end{verbatim}

\texttt{bgtz} opcode

\begin{verbatim}
000111
$9
01001
$0
00000
\end{verbatim}

(16 bits)

Offset -- to be determined
Assume that

- Address of the above instruction is 44 (decimal).
- Address of the label `end_loop` is 200 (decimal).

**Note:** When the above instruction in location 44 is being executed, the value of PC is 48. (Recall that PC is incremented right after instruction fetch.)

In PC-relative mode, the offset in bytes for label `end_loop` is given by

\[
\text{Address of } \text{end}\_\text{loop} - \text{PC value} = 200 - 48 = 152.
\]

- The offset in words = 152/4 = 38.
- This value (38) gets stored in the immediate operand field of the instruction.
Thus, the instruction

\texttt{bgtz \$9, end\_loop}

where the PC-relative offset of \texttt{end\_loop} is 38 words is stored as follows:

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
$9$ & $0$ & (16 bits) & \\
\hline
000111 & 01001 & 00000 & 0000000000100110 \\
\hline
\end{tabular}
\end{center}

\textbf{General Formula for Offset:}

\begin{align*}
\text{Offset in bytes} & = \text{Target address} - \text{Addr. of branch instruction} - 4 \\
\text{Offset in words} & = \frac{\text{Offset in bytes}}{4}
\end{align*}
Additional Notes Regarding Conditional Jumps

- If offset is negative, it is stored in 2’s complement form.
- Using PC-relative mode of addressing, the jump span is about $2^{15} = 32K$ words on either side of the jump instruction.

- Offset of 16 bits is adequate for most targets in a typical program.
- For larger offsets, instructions can be synthesized differently.
**Example:** Suppose we want to synthesize the instruction

\[
\text{bgtz } \$5, \text{ label\_too\_far}
\]

where the offset for *label\_too\_far* requires more than 16 bits.

We can realize the above instruction as follows:

\[
\text{blez } \$5, \text{ next}
\]

\[
\text{j } \text{ label\_too\_far}
\]

**next:**

The offset for the label *next* is only 4 bytes (or 1 word).
Problem 1: Show the machine representation for the TAL instruction

addi $5, $4, -49

Solution: The addi instruction uses the I-format. The opcode for addi is 001000.

+49 (in 16 bits): 0000 0000 0011 0001
-49 (2’s compl): 1111 1111 1100 1111

The required machine representation is:

$$\begin{array}{cccc}
6 & 5 & 5 & 16 \\
001000 & 00100 & 00101 & 1111111111001111
\end{array}$$

addi opcode

$4$ $5$ $-49$ (in 2’s compl.)
Problem 2: Find the machine representation for the TAL instruction

\[
\text{bgtz } 9, \text{ check_val}
\]

assuming that the address of the above instruction is 340 (decimal) and that the address of the label check_val is 128 (decimal).

Solution: This instruction also uses the I-format. The opcode for bgtz is 000111.

Offset calculation: Since the conditional jump instruction uses PC-relative addressing, the offset in bytes for the target check_val is

\[
128 - 340 - 4 = -216.
\]

The offset in words = \(-216/4 = -54\).
Solution to Problem 2 (continued)

+54 (in 16 bits): 0000 0000 0011 0110
-54 (2’s compl): 1111 1111 1100 1010

The required machine representation is:

$$\begin{array}{cccc}
6 & 5 & 5 & 16 \\
000111 & 01001 & 00000 & 1111111111001010 \\
\$9 & \$0 & -54 (2's compl.) \\
\text{bgtz opcode} & & & \\
\end{array}$$