## MIPS Instructions

Note: You can have this handout on both exams.

### Instruction Formats:

Instruction formats: all 32 bits wide (one word):

<table>
<thead>
<tr>
<th></th>
<th>6</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type format</td>
<td>Op-code</td>
<td>R&lt;sub&gt;s&lt;/sub&gt;</td>
<td>R&lt;sub&gt;t&lt;/sub&gt;</td>
<td>R&lt;sub&gt;d&lt;/sub&gt;</td>
<td>SA</td>
<td>Funct-code</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>I-type format</td>
<td>Op-code</td>
<td>R&lt;sub&gt;s&lt;/sub&gt;</td>
<td>R&lt;sub&gt;t&lt;/sub&gt;</td>
<td>2’s complement constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J-type format</td>
<td>Op-code</td>
<td>jump_target</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>^</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 31</td>
<td>bit 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Instructions and their formats

**General notes:**

a. R<sub>s</sub>, R<sub>t</sub>, and R<sub>d</sub> specify general purpose registers
b. Square brackets ([]) indicate “the contents of”
c. [PC] specifies the address of the instruction in execution
d. I specifies part of instruction and its subscripts indicate bit positions of sub-fields
e. || indicates concatenation of bit fields
f. Superscripts indicate repetition of a binary value
g. M<i> is a value (contents) of the word beginning at the memory address i
h. m<i> is a value (contents) of the byte at the memory address i
i. all integers are in 2’s complement representation if not indicated as unsigned

1. **addition with overflow:** **add** instruction

<table>
<thead>
<tr>
<th></th>
<th>000000</th>
<th>R&lt;sub&gt;s&lt;/sub&gt;</th>
<th>R&lt;sub&gt;t&lt;/sub&gt;</th>
<th>R&lt;sub&gt;d&lt;/sub&gt;</th>
<th>00000</th>
<th>100000</th>
</tr>
</thead>
</table>

Effects of the instruction: R<sub>d</sub> ← [R<sub>s</sub>] + [R<sub>t</sub>]; PC ← [PC] + 4
(If overflow then exception processing)

Assembly format: **add R<sub>d</sub>,R<sub>s</sub>,R<sub>t</sub>**
2. add without overflow: **addu** instruction
   Identical as **add** instruction, except:
   - funct = 33_{dec}
   - overflow ignored

3. subtract with overflow: **sub** instruction

   R-type format
   +--------+-------+------+-------+-------+--------+
   | 000000 |   R_s |   R_t |   R_d | 00000 | 100010 |
   +--------+-------+------+-------+-------+--------+

   Effects of the instruction: R_d <-- [R_s] - [R_t]; PC <-- [PC] + 4
   (If overflow then exception processing)
   Assembly format: **sub** R_d, R_s, R_t

4. subtract without overflow: **subu** instruction
   Identical as **sub** instruction, except:
   - funct = 35_{dec}
   - overflow ignored

5. multiply: **mul** instruction

   R-type format
   +--------+-------+-------+-------+-------+--------+
   | 000000 |   R_s |   R_t | 00000 | 00000 | 011000 |
   +--------+-------+-------+-------+-------+--------+

   Effects of the instruction: Hi||Lo <-- [R_s] * [R_t]; PC <-- [PC] + 4
   Assembly format: **mul** R_s, R_t

6. unsigned multiply: **mulu** instruction
   Identical as **mul** instruction, except:
   - funct = 25_{dec}
   - contents of R_s and R_t are considered as unsigned integers

7. divide: **div** instruction

   R-type format
   +--------+-------+-------+-------+-------+--------+
   | 000000 |   R_s |   R_t | 00000 | 00000 | 011010 |
   +--------+-------+-------+-------+-------+--------+

   Effects of the instruction: Lo <-- [R_s] / [R_t]; Hi <-- [R_s]mod[R_t]
   PC <-- [PC] + 4
   Assembly format: **div** R_s, R_t

8. unsigned divide: **divu** instruction
   Identical as **div** instruction, except:
   - funct = 27_{dec}
   - contents of R_s and R_t are considered as unsigned integers
9. set less than: **slt** instruction  

**R-type format**  

| 000000 | R_s | R_t | R_d | 00000 | 101010 |

Effects of the instruction:  
if \([R_s] < [R_t]\) then \(R_d \leftarrow 0^{31} \mid 1\) else \(R_d \leftarrow 0^{32}\); PC \(\leftarrow [PC] + 4\)  
Assembly format: **slt** \(R_d, R_s, R_t\)

10. set less than unsigned: **sltu** instruction  
Identical as **slt** instruction, except:  
- funct = 43\(\text{dec}\)  
- contents of \(R_s\) and \(R_t\) are considered as unsigned integers.

11. logical and: **and** instruction  

**R-type format**  

| 000000 | R_s | R_t | R_d | 00000 | 100100 |

Effects of the instruction:  \(R_d \leftarrow [R_s] \text{ AND } [R_t]\); PC \(\leftarrow [PC] + 4\)  
Assembly format: **and** \(R_d, R_s, R_t\)

12 - 14. logical or, nor & exclusive or: **or**, **nor**, & **xor** instructions  
Identical as **and** instruction, except:  
- funct=37\(\text{dec}\) for **or** instruction  
- funct=39\(\text{dec}\) for **nor** instruction  
- funct=40\(\text{dec}\) for **xor** instruction  
- appropriate logical function performed instead of logical **and**

15. addition immediate with overflow: **addi** instruction  

**I-type format**  

| 001000 | R_s | R_t | immediate |  

Effects of the instruction:  
\(R_t \leftarrow [R_s] + ([I_{16}]^{16} \mid [I_{15..0}]);\) PC \(\leftarrow [PC] + 4\)  
(If overflow then exception processing)  
Assembly format: **addi** \(R_t, R_s,\text{ immediate}\)

16. addition immediate without overflow: **addiu** instruction  
Identical as **addi** instruction, except:  
- op-code=9\(\text{dec}\)  
- overflow ignored
17. set less than immediate: \texttt{slti} instruction

I-type format: 

\begin{verbatim}
+--------+-------+-------+-----------------------+
| 001010 | R_s  | R_t  | immediate            |
|--------+-------+-------+-----------------------+
\end{verbatim}

Effects of the instruction:

\[
\begin{align*}
\text{if } [R_s] & < ([I_{11}]_{16} | [I_{15..0}]) \text{ then } R_t &\leftarrow 0^{31} | 1 \\
\text{else } R_t &\leftarrow 0^{32} \\
\text{PC } &\leftarrow [PC] + 4
\end{align*}
\]

Assembly format: \texttt{slti} \texttt{R_t,R_s,immediate}

18. set less than immediate unsigned: \texttt{sltiu} instruction

Identical as \texttt{slti} instruction, except:

- op-code = 11_{\text{dec}}
- contents in the comparison are considered as unsigned integers.

19. logical and immediate: \texttt{andi} instruction

I-type format: 

\begin{verbatim}
+--------+-------+-------+-----------------------+
| 001100 | R_s  | R_t  | immediate            |
|--------+-------+-------+-----------------------+
\end{verbatim}

Effects of the instruction: 

\[
R_t \leftarrow [R_s] \text{ AND } (0^{16} | [I_{15..0}]);
\]

\[
\text{PC } \leftarrow [PC] + 4
\]

Assembly format: \texttt{andi} \texttt{R_t,R_s,immediate}

20-21. logical or immediate & xor immediate: \texttt{ori}, & \texttt{xori} instr.

Identical as \texttt{andi} instruction, except:

- op-code=13_{\text{dec}} for \texttt{ori} instruction
- op-code=14_{\text{dec}} for \texttt{xori} instruction
- appropriate logical function performed instead of logical AND

22. load word: \texttt{lw} instruction

I-type format: 

\begin{verbatim}
+--------+-------+-------+-----------------------+
| 100011 | R_s  | R_t  | offset               |
|--------+-------+-------+-----------------------+
\end{verbatim}

Effects of the instruction: 

\[
\begin{align*}
\text{if an illegal memory address then exception processing}
\end{align*}
\]

Assembly format: \texttt{lw} \texttt{R_t,offset(R_s)}

23. store word: \texttt{sw} instruction

I-type format: 

\begin{verbatim}
+--------+-------+-------+-----------------------+
| 101011 | R_s  | R_t  | offset               |
|--------+-------+-------+-----------------------+
\end{verbatim}

Effects of the instruction: 

\[
\begin{align*}
\text{if an illegal memory address then exception processing}
\end{align*}
\]

Assembly format: \texttt{sw} \texttt{R_t,offset(R_s)}
24. load unsigned byte: **lbu** instruction

I-type format: | 100100 | Rs | Rt | offset |

Effects of the instruction:

\[ R_t \leftarrow 0^{24} \mid m\{[R_s] + [I_{15}]^{16} \mid [I_{15..0}]\} \]

(PC \leftarrow [PC] + 4)

*(If an illegal memory address then exception processing)*

Assembly format: **lbu R_s,offset(R_t)**

---

25. load byte: **lb** instruction

Identical as **lbu** instruction, except:
- leftmost 24 bits of \( R_t \) are loaded by a value of leftmost bit of the byte instead of zeros
- op-code = \( 32_{\text{dec}} \)

---

26. store byte: **sb** instruction

I-type format: | 101000 | Rs | Rt | offset |

Effects of the instruction:

\[ m\{[R_s] + [I_{15}]^{16} \mid [I_{15..0}]\} \leftarrow [R_t]_{7..0} \]

(PC \leftarrow [PC] + 4)

*(If an illegal memory address then exception processing)*

Assembly format: **sb R_s,offset(R_t)**

---

27. load upper immediate: **lui** instruction

I-type format: | 001111 | 00000 | Rt | immediate |

Effects of the instruction:

\[ R_t \leftarrow [I_{15..0}] \mid 0^{16}; \text{PC} \leftarrow [PC] + 4 \]

Assembly format: **lui R_t,immediate**

---

28. branch on equal: **beq** instruction

I-type format: | 000100 | Rs | Rt | offset |

Effects of the instruction:

if \([R_s] = [R_t]\) then \( \text{PC} \leftarrow [PC] + 4 + ([I_{15}]^{14} \mid [I_{15..0}] \mid 0^5) \)

(i.e. \( \text{PC} \leftarrow [PC] + 4 + 4 \times \text{offset} \))

else \( \text{PC} \leftarrow [PC] + 4 \)

Assembly format: **beq R_s,R_t,offset**
29. branch on not equal: bne instruction

I-type format: | 000101 | R_s | R_t | offset |

Effects of the instruction:
if [R_s] <> [R_t] then PC <-- [PC] + 4 + ([I_15] || [I_15..0] || 0^2)
else PC <-- [PC] + 4
Assembly format: bne R_s,R_t,offset

30. branch on less than or equal zero: blez instruction

I-type format: | 000110 | R_s | 00000 | offset |

Effects of the instruction:
if [R_s] < 0 then PC <-- [PC] + 4 + ([I_15] || [I_15..0] || 0^2)
else PC <-- [PC] + 4
Assembly format: blez R_s,offset

31. branch on greater than zero: bgtz instruction

I-type format: | 000111 | R_s | 00000 | offset |

Effects of the instruction:
if [R_s] > 0 then PC <-- [PC] + 4 + ([I_15] || [I_15..0] || 0^2)
else PC <-- [PC] + 4
Assembly format: bgtz R_s,offset

32. branch on less than zero: bltz instruction

I-type format: | 000001 | R_s | 00000 | offset |

Effects of the instruction:
if [R_s] < 0 then PC <-- [PC] + 4 + ([I_15] || [I_15..0] || 0^2)
else PC <-- [PC] + 4
Assembly format: bltz R_s,offset

33. jump: j instruction

J-type format | 000010 | jump_target |

Effects of the instruction: PC <-- [PC_{31..28}] || [I_{25..0}] || 0^2
Assembly format: j jump_target
34. **jump and link**: `jal` instruction

J-type format

| 000011 | jump_target |

Effects of the instruction: \( R_{s} \leftarrow [PC] + 4 \)
\( PC \leftarrow [PC_{31..28}] || [I_{25..0}] || 0^2 \)
Assembly format: `jal jump_target`

35. **jump register**: `jr` instruction

R-type format

| 000000 | \( R_{s} \) | 00000 | 00000 | 00000 | 001000 |

Effects of the instruction: \( PC \leftarrow [R_{s}] \)
Assembly format: `jr R_{s}`

36. **jump and link register**: `jalr` instruction

R-type format

| 000000 | \( R_{s} \) | 00000 | 00000 | \( R_{d} \) | 00000 | 001001 |

Effects of the instruction: \( R_{d} \leftarrow [PC] + 4; \ PC \leftarrow [R_{s}] \)
Assembly format: `jalr R_{d},R_{s}`

37. **no operation**: `nop` instruction

R-type format

| 000000 | 00000 | 00000 | 00000 | 00000 | 00000 | 00000 |

Effects of the instruction: \( PC \leftarrow [PC] + 4 \)
Assembly format: `nop` (= `sll R_{s},0` shift logical left 0)

38. **move from Hi**: `mfhi` instruction

R-type format

| 000000 | 00000 | 00000 | \( R_{d} \) | 00000 | 010000 |

Effects of the instruction: \( R_{d} \leftarrow [Hi]; \ PC \leftarrow [PC] + 4 \)
Assembly format: `mfhi R_{d}`

39. **move from Lo**: `mflo` instruction

R-type format

| 000000 | 00000 | 00000 | \( R_{d} \) | 00000 | 010010 |

Effects of the instruction: \( R_{d} \leftarrow [Lo]; \ PC \leftarrow [PC] + 4 \)
Assembly format: `mflo R_{d}`
Exception Handling
When a condition for any exception (overflow, illegal op-code, division by zero, etc.) occurs the following hardware exception processing is performed:

\[
\begin{align*}
\text{EPC} & \leftarrow [\text{PC}] \\
\text{Cause}_\text{Reg} & \leftarrow \begin{cases} 
0^{28} || 1010 & \text{if illegal op-code (10)} \\
0^{28} || 1100 & \text{if overflow (12)} \\
0^{29} || 100 & \text{if illegal memory address (4)} \\
& \text{etc.}
\end{cases} \\
\text{PC} & \leftarrow 8000180_{\text{hex}}
\end{align*}
\]

40. move from EPC: \texttt{mfepc} instruction

\[
\begin{array}{cccccccc}
\text{R-type format} & 010000 & 00000 & R_t & 01110 & 00000 & 000000 & \\
\end{array}
\]

Effects of the instruction: \(R_d \leftarrow \text{[EPC]}; \: \text{PC} \leftarrow \text{[PC]} + 4\)

Assembly format: \texttt{mfepc R_t} (This is mfc0 Rt,CP0reg14)

41. move from Cause_Reg: \texttt{mfco} instruction

\[
\begin{array}{cccccccc}
\text{R-type format} & 010000 & 00000 & R_s & 01101 & 00000 & 000000 & \\
\end{array}
\]

Effects of the instruction: \(R_d \leftarrow \text{[Cause_Reg]}; \: \text{PC} \leftarrow \text{[PC]} + 4\)

Assembly format: \texttt{mfco R_s} (This is mfc0 Rt,CP0reg13)

Floating Point Instructions

42. load word into co-processor 1: \texttt{lwcl} instruction

\[
\begin{array}{cccccccc}
\text{I-type format:} & 110001 & R_s & f_c & \text{offset} & \\
\end{array}
\]

Effects of the instruction: \(f_c \leftarrow M[R_s] + \lfloor I_{15} \rfloor_{16} || \lfloor I_{15..0} \rfloor\)

Assembly format: \texttt{lwcl f_c,offset(R_s)}

43. store word from co-processor 1: \texttt{swcl} instruction

\[
\begin{array}{cccccccc}
\text{I-type format:} & 111001 & R_s & f_c & \text{offset} & \\
\end{array}
\]

Effects of the instruction: \(M[R_s] + \lfloor I_{15} \rfloor_{16} || \lfloor I_{15..0} \rfloor \leftarrow f_c\)

Assembly format: \texttt{swcl f_c,offset(R_s)}
44. addition single precision: \texttt{add.s} instruction

\begin{verbatim}
+-----------------+-------+-------+-------+-------+
| \text{R-type format} | 010001 | 00000 | \text{f}_t | \text{f}_s | \text{f}_d |
\end{verbatim}

Effects of the instruction: \( \text{f}_d \leftarrow [\text{f}_s] + [\text{f}_t]; \) \( \text{PC} \leftarrow [\text{PC}] + 4 \)
(If overflow then exception processing)

Assembly format: \texttt{add.s R_s,R_s,R_t}

45. addition double precision: \texttt{add.d} instruction

\begin{verbatim}
+-----------------+-------+-------+-------+-------+-------+
| \text{R-type format} | 010001 | 00001 | \text{f}_t | \text{f}_s | \text{f}_d |
\end{verbatim}

Effects of the instruction: \( \text{f}_{d,1} \leftarrow [\text{f}_s] + [\text{f}_t] + [f_{s,1}] + [f_{t,1}]; \) \( \text{PC} \leftarrow [\text{PC}] + 4 \)
(If overflow then exception processing)

Assembly format: \texttt{add.d f_s,f_s,f_t,f_t,}

45. subtract single precision: \texttt{sub.s} instruction

Similar as \texttt{add.s} but with \texttt{funct}=1

46. subtract double precision: \texttt{sub.d} instruction

Similar as \texttt{add.d} but with \texttt{funct}=1

47. multiply single precision: \texttt{mul.s} instruction

Similar as \texttt{add.s} but with \texttt{funct}=2

48. multiply double precision: \texttt{mul.d} instruction

Similar as \texttt{add.d} but with \texttt{funct}=2

49. divide single precision: \texttt{div.s} instruction

Similar as \texttt{add.s} but with \texttt{funct}=3

50. divide double precision: \texttt{div.d} instruction

Similar as \texttt{add.d} but with \texttt{funct}=3