

## **General Instructions**

- You can download all required source files, that are provided as starting points for the following exercises, from the course website.
- Each group should submit a zip (or tar.gz) file containing all the necessary source code files via the course website.
- Each group should submit a report, written in French or English, in the **PDF** file format via the course website.
- Each student has to be member of a group, where groups should generally consist of 3 students.

# 1 MIPS Instruction Set

**Aims:** *Understand the instruction set architecture and encoding of the MIPS processor.*

Consider the following MIPS program represented the binary code of a simple function:

```
0:  1080000d
4:  00000000
8:  80850000
c:  00000000
10: 10a00007
14: 00001025
18: 24840001
1c: 38a30020
20: 80850000
24: 0003182b
28: 14a0ffffb
2c: 00431021
30: 03e00008
34: 00000000
38: 03e00008
3c: 2402fffff
```

- Use the attached cheat sheet (at the end of the assignment) in order to determine which MIPS instructions appear in the program. Determine the instruction format for each instruction.
- Determine the operands for each instruction. For registers determine both, the register number and the symbolic register name.
- There are conditional branches in the function. Determine to which instructions they branch.
- What is the function actually doing? What is its return value?

## 2 MIPS Tool Chain

**Aims:** *Understand the interplay between compiler and computer architecture.*

- Write a C program matching the program from above.
- Compile the program using the MIPS compiler installed on the lab machines using the following command line:

```
mips-linux-gnu-gcc -mips1 -c -g -o mips-prog.o <input-file>
```

- Disassemble the compiled program (`mips-prog.o`) with the `objdump` tool using the following command line:

```
mips-linux-gnu-objdump -d mips-prog.o
```

- Compare the resulting assembly code obtained from the `objdump` tool with the code from above. Explain why the code looks so differently? The code contains many `lw` instructions that use the register `s8`. What is the purpose of register `s8`? What are these memory access instructions doing?
- Try to change the compiler options (enable/disable optimizations using the option `-O0`, `-O`, or `-O3`) and see how this changes the code that you can see using the `objdump` tool.

### 3 MIPS Architecture

**Aims:** *Understand MIPS program execution on a pipelined processor.*

For the following exercises assume a MIPS implementation as discussed in the lecture:

- The pipeline consists of 5 stages (IF, ID, EX, MEM, WB).
- Registers are read in the ID stage and written in the WB stage.
- Memory accesses are performed in the MEM stage.
  - The address computation is performed in the EX stage.
  - Assume that memory accesses take two cycle in the MEM stage, i.e., every memory access induces a stall cycle.
  - Data hazards between a memory load (in the MEM stage) and another instruction immediately using its results (in the EX stage) are ignored by the processor.
- Branches are performed in the EX stage.
  - The instruction following a branch is executed and never flushed.
  - The second instruction after a branch is flushed if the branch is taken.
- For arithmetic instructions forwarding is performed as explained in the lecture.

#### 3.1 Program Flow

- Given the program from Question 1, provide a list of instructions that are executed along with a brief explanation of the processor/program state.

Assume that the program starts with the following initial processor state:

- Register a0 has the value 0x200.
- All other registers have the value zero (0x0).
- The memory contents at the address range 0x200 through 0x203 is given as follows:

| Address | Value |
|---------|-------|
| 0x200   | 0x61  |
| 0x201   | 0x20  |
| 0x202   | 0x62  |
| 0x203   | 0x0   |

- All other memory cells have a value of zero (0x0).

Provide a full list of instructions until the function terminates by executing a `jx` instruction. You can follow the example below:

| PC  | Instruction          | a0    | a1  | v0  | v1  | Explanation   |
|-----|----------------------|-------|-----|-----|-----|---|
| 0x0 | <b>beqz</b> a0, 0x38 | 0x200 | 0x0 | 0x0 | 0x0 | Check if register a0 is zero; result: false (non-taken) |
| 0x4 | <b>nop</b>           | 0x200 | 0x0 | 0x0 | 0x0 | No change   |
| 0x8 | <b>lb</b> a1, 0(a0)  | ...   | ... | ... | ... | ...   |

### 3.2 Pipeline Diagram

- Draw a pipeline diagram showing all the instructions executed by the function as determined above. Assume a processor implementation as described above. Highlight all forms of hazards that occur and graphically distinguish resolution mechanisms (e.g., forwarding, stalls, flushing).

## 4 Processor Design

**Aims:** *Explain and understand the instruction set of a processor and its implementation using a simple pipeline.*

### 4.1 Instruction Set Architecture

Describe the instruction set and the binary representation of the instructions of a simple processor. Your processor should respect the following list of characteristics:

- All instructions should be encoded in 16 bits. Apart from the instruction width, you are free to define the binary format yourself.
- Your processor should have 16 registers, i.e., encoding a register operand requires 4-bits. Assume that each register is 32-bit wide.
- The PC of your processor should be 32-bit wide.
- Your processor has separate instruction and data memories.
- Define at least three different arithmetic/logic instructions operating on 3 register operands (reading 2 registers and writing 1).
- Define an instruction to read an 8-bit value from data memory (load). The instruction should take two register operands (reading 1 and writing 1) and a 5-bit immediate operand. The address used to access the memory is derived by adding the value of the read register to the immediate.
- Define an instruction to write an 8-bit value to data memory (store). The instruction should take two register operands (reading 2) and a 5-bit immediate operand. The address computation is the same as for loads.
- Define an instruction to copy an immediate value into a register. You may choose whether the value is sign-extended or not.
- Define a conditional branch instruction having 1 register operand (read) and a signed 10-bit immediate operand. The branch is taken when the register operand is non-zero. The new PC value is then computed as follows:  $PC_{new} = PC_{old} + imm$ . Untaken branches simply continue straight.
- Define an unconditional jump instruction having 1 register operand (read). The new PC value is obtained by copying the register operand's value into the PC register. The jump is always taken.

Using the instruction set you just defined, please complete the following exercises and include your replies in the report:

- Describe each instruction of your processor. Explain what the instruction is doing, how it can be written in human readable form (assembly), and how it is encoded in binary form.

- Group instructions into binary formats, similar to the I-, J-, and R-format discussed for MIPS in the lecture. Illustrate the formats using figures in your report.
- Provide a sequence of instructions in assembly form that allows to load the constant 65534 into a register using the instructions of your processor. Give a short explanation of each instruction and each intermediate result of your code.
- Translate the C-code from Question 1 to corresponding instructions of your processor. Assume that the input pointer is provided in the second register of your processor and that the result should be returned in the first register. The return address is similarly provided in register 15. Try to use the instructions of your processor as optimal as possible in order to minimize the number of instructions. There is no need to preserve any register values in your code, i.e., you can overwrite any register if needed.

## 4.2 Pipelining

Now define the pipeline of your processor, while respecting the following characteristics:

- Your processor should have three pipeline stages: instruction fetch (**IF**), instruction de-code (**ID**), and execute (**EX**).
- For arithmetic the three pipeline stages correspond, except for minor differences, to the pipeline stages of the MIPS processor discussed in the lecture.
- Memory accesses are, however, different. Since no address computation is needed (the address is simply the value of a register operand), the memory access can be executed in the **EX** stage.
- Conditional branches and unconditional jumps should be executed in the **ID** stage. Taken branches/jumps thus have to flush a single instructions in the **IF** stage only.
- Assume that the processor registers are written at the beginning of the **EX** stage and read at the end of the **ID** stage, i.e., values written in the **EX** stage are immediately available in the **ID** stage.

Using the instruction set of your processor from the previous exercise and your pipeline design, please complete the following exercises and include your replies in the report:

- Draw a diagram of your processor's design. Use registers, pipeline registers, multiplexers, ALUs, . . . , as you need them. You can ignore the logic needed to flushing the **IF** stages for conditional branches. Describe relevant parts of the diagram.
- Which kinds of hazards (data, control, or structural) can you encounter for your processor? Explain under which circumstances these hazards occur. How are these hazards resolved?
- Does your processor need *forwarding* (as discussed in the lecture) for the instructions in the **EX** stage? What about the conditional branch that is executed in the **ID** stage? Explain for both cases why it is needed or why it is not needed.

# MIPS Reference Data

①



## CORE INSTRUCTION SET

| NAME, MNEMONIC              | FOR-MAT | OPERATION (in Verilog)   | OPCODE / FUNCT (Hex)    |
|-----------------------------|---------|--|-------------------------|
| Add                         | add R   | $R[rd] = R[rs] + R[rt]$  | (1) 0/20 <sub>hex</sub> |
| Add Immediate               | addi I  | $R[rt] = R[rs] + \text{SignExtImm}$  | (1,2) 8 <sub>hex</sub>  |
| Add Imm. Unsigned           | addiu I | $R[rt] = R[rs] + \text{SignExtImm}$  | (2) 9 <sub>hex</sub>    |
| Add Unsigned                | addu R  | $R[rd] = R[rs] + R[rt]$  | 0/21 <sub>hex</sub>     |
| And                         | and R   | $R[rd] = R[rs] \& R[rt]$   | 0/24 <sub>hex</sub>     |
| And Immediate               | andi I  | $R[rt] = R[rs] \& \text{ZeroExtImm}$   | (3) c <sub>hex</sub>    |
| Branch On Equal             | beq I   | if( $R[rs] == R[rt]$ )<br>$PC = PC + 4 + \text{BranchAddr}$                  | (4) 4 <sub>hex</sub>    |
| Branch On Not Equal         | bne I   | if( $R[rs] != R[rt]$ )<br>$PC = PC + 4 + \text{BranchAddr}$                  | (4) 5 <sub>hex</sub>    |
| Jump                        | j J     | $PC = \text{JumpAddr}$   | (5) 2 <sub>hex</sub>    |
| Jump And Link               | jal J   | $R[31] = PC + 8; PC = \text{JumpAddr}$                                       | (5) 3 <sub>hex</sub>    |
| Jump Register               | jr R    | $PC = R[rs]$   | 0/08 <sub>hex</sub>     |
| Load Byte Unsigned          | lbu I   | $R[rt] = \{24'b0, M[R[rs] + \text{SignExtImm}](7:0)\}$                       | (2) 24 <sub>hex</sub>   |
| Load Halfword Unsigned      | lhu I   | $R[rt] = \{16'b0, M[R[rs] + \text{SignExtImm}](15:0)\}$                      | (2) 25 <sub>hex</sub>   |
| Load Linked                 | ll I    | $R[rt] = M[R[rs] + \text{SignExtImm}]$                                       | (2,7) 30 <sub>hex</sub> |
| Load Upper Imm.             | lui I   | $R[rt] = \{\text{imm}, 16'b0\}$  | f <sub>hex</sub>        |
| Load Word                   | lw I    | $R[rt] = M[R[rs] + \text{SignExtImm}]$                                       | (2) 23 <sub>hex</sub>   |
| Nor                         | nor R   | $R[rd] = \sim (R[rs]   R[rt])$   | 0/27 <sub>hex</sub>     |
| Or                          | or R    | $R[rd] = R[rs]   R[rt]$  | 0/25 <sub>hex</sub>     |
| Or Immediate                | ori I   | $R[rt] = R[rs]   \text{ZeroExtImm}$  | (3) d <sub>hex</sub>    |
| Set Less Than               | slt R   | $R[rd] = (R[rs] < R[rt]) ? 1 : 0$  | 0/2a <sub>hex</sub>     |
| Set Less Than Imm.          | slti I  | $R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$                                | (2) a <sub>hex</sub>    |
| Set Less Than Imm. Unsigned | sltiu I | $R[rt] = (R[rs] < \text{SignExtImm}) ? 1 : 0$                                | (2,6) b <sub>hex</sub>  |
| Set Less Than Unsig.        | sltu R  | $R[rd] = (R[rs] < R[rt]) ? 1 : 0$  | (6) 0/2b <sub>hex</sub> |
| Shift Left Logical          | sll R   | $R[rd] = R[rt] \ll \text{shamt}$   | 0/00 <sub>hex</sub>     |
| Shift Right Logical         | srl R   | $R[rd] = R[rt] \gg \text{shamt}$   | 0/02 <sub>hex</sub>     |
| Store Byte                  | sb I    | $M[R[rs] + \text{SignExtImm}](7:0) = R[rt](7:0)$                             | (2) 28 <sub>hex</sub>   |
| Store Conditional           | sc I    | $M[R[rs] + \text{SignExtImm}] = R[rt];$<br>$R[rt] = (\text{atomic}) ? 1 : 0$ | (2,7) 38 <sub>hex</sub> |
| Store Halfword              | sh I    | $M[R[rs] + \text{SignExtImm}](15:0) = R[rt](15:0)$                           | (2) 29 <sub>hex</sub>   |
| Store Word                  | sw I    | $M[R[rs] + \text{SignExtImm}] = R[rt]$                                       | (2) 2b <sub>hex</sub>   |
| Subtract                    | sub R   | $R[rd] = R[rs] - R[rt]$  | (1) 0/22 <sub>hex</sub> |
| Subtract Unsigned           | subu R  | $R[rd] = R[rs] - R[rt]$  | 0/23 <sub>hex</sub>     |

- (1) May cause overflow exception
- (2)  $\text{SignExtImm} = \{16\{\text{immediate}[15]\}, \text{immediate}\}$
- (3)  $\text{ZeroExtImm} = \{16\{1b'0\}, \text{immediate}\}$
- (4)  $\text{BranchAddr} = \{14\{\text{immediate}[15]\}, \text{immediate}, 2'b0\}$
- (5)  $\text{JumpAddr} = \{PC + 4[31:28], \text{address}, 2'b0\}$
- (6) Operands considered unsigned numbers (vs. 2's comp.)
- (7) Atomic test&set pair;  $R[rt] = 1$  if pair atomic, 0 if not atomic

## BASIC INSTRUCTION FORMATS

|   |        |         |       |           |       |       |
|---|--------|---------|-------|-----------|-------|-------|
| R | opcode | rs      | rt    | rd        | shamt | funct |
|   | 31     | 26 25   | 21 20 | 16 15     | 11 10 | 6 5   |
| I | opcode | rs      | rt    | immediate |       |       |
|   | 31     | 26 25   | 21 20 | 16 15     |       |       |
| J | opcode | address |       |           |       |       |
|   | 31     | 26 25   |       |           |       |       |

## ARITHMETIC CORE INSTRUCTION SET

| NAME, MNEMONIC   | FOR-MAT   | OPERATION   | OPCODE / FUNCT (Hex) |
|--|-----------|---|----------------------|
| Branch On FP True  | bc1t FI   | if( $FPcond$ ) $PC = PC + 4 + \text{BranchAddr}$  | (4) 11/8/1--         |
| Branch On FP False   | bc1f FI   | if(! $FPcond$ ) $PC = PC + 4 + \text{BranchAddr}$                                       | (4) 11/8/0--         |
| Divide   | div R     | $Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$   | 0/--/1a              |
| Divide Unsigned  | divu R    | $Lo = R[rs]/R[rt]; Hi = R[rs]\%R[rt]$   | (6) 0/--/1b          |
| FP Add Single  | add.s FR  | $F[fd] = F[fs] + F[ft]$   | 11/10/--/0           |
| FP Add Double  | add.d FR  | $\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} + \{F[ft], F[ft+1]\}$                          | 11/11/--/0           |
| FP Compare Single  | c.x.s* FR | $FPcond = (F[fs] \text{ op } F[ft]) ? 1 : 0$  | 11/10/--/y           |
| FP Compare Double  | c.x.d* FR | $FPcond = (\{F[fs], F[fs+1]\} \text{ op } \{F[ft], F[ft+1]\}) ? 1 : 0$                  | 11/11/--/y           |
| * (x is eq, lt, or le) (op is ==, <, or <=) (y is 32, 3c, or 3e) |           |   |                      |
| FP Divide Single   | div.s FR  | $F[fd] = F[fs] / F[ft]$   | 11/10/--/3           |
| FP Divide Double   | div.d FR  | $\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} / \{F[ft], F[ft+1]\}$                          | 11/11/--/3           |
| FP Multiply Single   | mul.s FR  | $F[fd] = F[fs] * F[ft]$   | 11/10/--/2           |
| FP Multiply Double   | mul.d FR  | $\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} * \{F[ft], F[ft+1]\}$                          | 11/11/--/2           |
| FP Subtract Single   | sub.s FR  | $F[fd] = F[fs] - F[ft]$   | 11/10/--/1           |
| FP Subtract Double   | sub.d FR  | $\{F[fd], F[fd+1]\} = \{F[fs], F[fs+1]\} - \{F[ft], F[ft+1]\}$                          | 11/11/--/1           |
| Load FP Single   | lwc1 I    | $F[rt] = M[R[rs] + \text{SignExtImm}]$  | (2) 31/--/--         |
| Load FP Double   | ldc1 I    | $F[rt] = M[R[rs] + \text{SignExtImm}];$<br>$F[rt+1] = M[R[rs] + \text{SignExtImm} + 4]$ | (2) 35/--/--         |
| Move From Hi   | mfhi R    | $R[rd] = Hi$  | 0/--/--/10           |
| Move From Lo   | mflo R    | $R[rd] = Lo$  | 0/--/--/12           |
| Move From Control  | mfc0 R    | $R[rd] = CR[rs]$  | 10/0/--/0            |
| Multiply   | mult R    | $\{Hi, Lo\} = R[rs] * R[rt]$  | 0/--/--/18           |
| Multiply Unsigned  | multu R   | $\{Hi, Lo\} = R[rs] * R[rt]$  | (6) 0/--/--/19       |
| Shift Right Arith.   | sra R     | $R[rd] = R[rt] \gg \text{shamt}$  | 0/--/--/3            |
| Store FP Single  | swc1 I    | $M[R[rs] + \text{SignExtImm}] = F[rt]$  | (2) 39/--/--         |
| Store FP Double  | sdc1 I    | $M[R[rs] + \text{SignExtImm}] = F[rt];$<br>$M[R[rs] + \text{SignExtImm} + 4] = F[rt+1]$ | (2) 3d/--/--         |

## FLOATING-POINT INSTRUCTION FORMATS

|    |        |       |       |           |       |       |
|----|--------|-------|-------|-----------|-------|-------|
| FR | opcode | fmt   | ft    | fs        | fd    | funct |
|    | 31     | 26 25 | 21 20 | 16 15     | 11 10 | 6 5   |
| FI | opcode | fmt   | ft    | immediate |       |       |
|    | 31     | 26 25 | 21 20 | 16 15     |       |       |

## PSEUDOINSTRUCTION SET

| NAME                         | MNEMONIC | OPERATION                                    |
|------------------------------|----------|--|
| Branch Less Than             | b1t      | if( $R[rs] < R[rt]$ ) $PC = \text{Label}$    |
| Branch Greater Than          | bgt      | if( $R[rs] > R[rt]$ ) $PC = \text{Label}$    |
| Branch Less Than or Equal    | b1e      | if( $R[rs] \leq R[rt]$ ) $PC = \text{Label}$ |
| Branch Greater Than or Equal | bge      | if( $R[rs] \geq R[rt]$ ) $PC = \text{Label}$ |
| Load Immediate               | li       | $R[rd] = \text{immediate}$                   |
| Move                         | move     | $R[rd] = R[rs]$                              |

## REGISTER NAME, NUMBER, USE, CALL CONVENTION

| NAME      | NUMBER | USE   | PRESERVED ACROSS A CALL? |
|-----------|--------|---|--------------------------|
| \$zero    | 0      | The Constant Value 0                                  | N.A.                     |
| \$at      | 1      | Assembler Temporary                                   | No                       |
| \$v0-\$v1 | 2-3    | Values for Function Results and Expression Evaluation | No                       |
| \$a0-\$a3 | 4-7    | Arguments   | No                       |
| \$t0-\$t7 | 8-15   | Temporaries   | No                       |
| \$s0-\$s7 | 16-23  | Saved Temporaries                                     | Yes                      |
| \$t8-\$t9 | 24-25  | Temporaries   | No                       |
| \$k0-\$k1 | 26-27  | Reserved for OS Kernel                                | No                       |
| \$gp      | 28     | Global Pointer  | Yes                      |
| \$sp      | 29     | Stack Pointer   | Yes                      |
| \$fp      | 30     | Frame Pointer   | Yes                      |
| \$ra      | 31     | Return Address  | Yes                      |



### OPCODES, BASE CONVERSION, ASCII SYMBOLS

| MIPS opcode<br>(31:26) | (1) MIPS funct<br>(5:0) | (2) MIPS funct<br>(5:0)         | Binary  | Decimal | Hexa-decimal | ASCII Character | Decimal | Hexa-decimal | ASCII Character |
|------------------------|-------------------------|---------------------------------|---------|---------|--------------|-----------------|---------|--------------|-----------------|
| (1)                    | sll                     | add <sub>f</sub>                | 00 0000 | 0       | 0            | NUL             | 64      | 40           | @               |
|                        |                         | sub <sub>f</sub>                | 00 0001 | 1       | 1            | SOH             | 65      | 41           | A               |
| j                      | srl                     | mul <sub>f</sub>                | 00 0010 | 2       | 2            | STX             | 66      | 42           | B               |
| jal                    | sra                     | div <sub>f</sub>                | 00 0011 | 3       | 3            | ETX             | 67      | 43           | C               |
| beq                    | sliv                    | sqr <sub>t</sub> <sub>f</sub>   | 00 0100 | 4       | 4            | EOT             | 68      | 44           | D               |
| bne                    |                         | abs <sub>f</sub>                | 00 0101 | 5       | 5            | ENQ             | 69      | 45           | E               |
| blez                   | srlv                    | mov <sub>f</sub>                | 00 0110 | 6       | 6            | ACK             | 70      | 46           | F               |
| bgtz                   | sra                     | neg <sub>f</sub>                | 00 0111 | 7       | 7            | BEL             | 71      | 47           | G               |
| addi                   | jr                      |                                 | 00 1000 | 8       | 8            | BS              | 72      | 48           | H               |
| addiu                  | jalr                    |                                 | 00 1001 | 9       | 9            | HT              | 73      | 49           | I               |
| slli                   | movz                    |                                 | 00 1010 | 10      | a            | LF              | 74      | 4a           | J               |
| sltiu                  | movn                    |                                 | 00 1011 | 11      | b            | VT              | 75      | 4b           | K               |
| andi                   | syscall                 | round.w <sub>f</sub>            | 00 1100 | 12      | c            | FF              | 76      | 4c           | L               |
| ori                    | break                   | trunc.w <sub>f</sub>            | 00 1101 | 13      | d            | CR              | 77      | 4d           | M               |
| xori                   |                         | ceil.w <sub>f</sub>             | 00 1110 | 14      | e            | SO              | 78      | 4e           | N               |
| lui                    | sync                    | floor.w <sub>f</sub>            | 00 1111 | 15      | f            | SI              | 79      | 4f           | O               |
| (2)                    |                         |                                 | 01 0000 | 16      | 10           | DLE             | 80      | 50           | P               |
|                        | mthi                    |                                 | 01 0001 | 17      | 11           | DC1             | 81      | 51           | Q               |
|                        | mflo                    | movz <sub>f</sub>               | 01 0010 | 18      | 12           | DC2             | 82      | 52           | R               |
|                        | mtlo                    | movn <sub>f</sub>               | 01 0011 | 19      | 13           | DC3             | 83      | 53           | S               |
|                        |                         |                                 | 01 0100 | 20      | 14           | DC4             | 84      | 54           | T               |
|                        |                         |                                 | 01 0101 | 21      | 15           | NAK             | 85      | 55           | U               |
|                        |                         |                                 | 01 0110 | 22      | 16           | SYN             | 86      | 56           | V               |
|                        |                         |                                 | 01 0111 | 23      | 17           | ETB             | 87      | 57           | W               |
|                        | mult                    |                                 | 01 1000 | 24      | 18           | CAN             | 88      | 58           | X               |
|                        | multu                   |                                 | 01 1001 | 25      | 19           | EM              | 89      | 59           | Y               |
|                        | div                     |                                 | 01 1010 | 26      | 1a           | SUB             | 90      | 5a           | Z               |
|                        | divu                    |                                 | 01 1011 | 27      | 1b           | ESC             | 91      | 5b           | [               |
|                        |                         |                                 | 01 1100 | 28      | 1c           | FS              | 92      | 5c           | \               |
|                        |                         |                                 | 01 1101 | 29      | 1d           | GS              | 93      | 5d           | ]               |
|                        |                         |                                 | 01 1110 | 30      | 1e           | RS              | 94      | 5e           | ^               |
|                        |                         |                                 | 01 1111 | 31      | 1f           | US              | 95      | 5f           | _               |
| lb                     | add                     | cvt.s <sub>f</sub>              | 10 0000 | 32      | 20           | Space           | 96      | 60           | `               |
| lh                     | addu                    | cvt.d <sub>f</sub>              | 10 0001 | 33      | 21           | !               | 97      | 61           | a               |
| lwl                    | sub                     |                                 | 10 0010 | 34      | 22           | "               | 98      | 62           | b               |
| lw                     | subu                    |                                 | 10 0011 | 35      | 23           | #               | 99      | 63           | c               |
| lbu                    | and                     | cvt.w <sub>f</sub>              | 10 0100 | 36      | 24           | \$              | 100     | 64           | d               |
| lhu                    | or                      |                                 | 10 0101 | 37      | 25           | %               | 101     | 65           | e               |
| lwr                    | xor                     |                                 | 10 0110 | 38      | 26           | &               | 102     | 66           | f               |
|                        | nor                     |                                 | 10 0111 | 39      | 27           | '               | 103     | 67           | g               |
| sb                     |                         |                                 | 10 1000 | 40      | 28           | (               | 104     | 68           | h               |
| sh                     |                         |                                 | 10 1001 | 41      | 29           | )               | 105     | 69           | i               |
| swl                    | sll                     |                                 | 10 1010 | 42      | 2a           | *               | 106     | 6a           | j               |
| sw                     | slltu                   |                                 | 10 1011 | 43      | 2b           | +               | 107     | 6b           | k               |
|                        |                         |                                 | 10 1100 | 44      | 2c           | ,               | 108     | 6c           | l               |
|                        |                         |                                 | 10 1101 | 45      | 2d           | -               | 109     | 6d           | m               |
|                        |                         |                                 | 10 1110 | 46      | 2e           | .               | 110     | 6e           | n               |
| swr                    |                         |                                 | 10 1111 | 47      | 2f           | /               | 111     | 6f           | o               |
| cache                  |                         |                                 |         |         |              |                 |         |              |                 |
| ll                     | tge                     | c.f <sub>f</sub>                | 11 0000 | 48      | 30           | 0               | 112     | 70           | p               |
| lwc1                   | tgeu                    | c.un <sub>f</sub>               | 11 0001 | 49      | 31           | 1               | 113     | 71           | q               |
| lwc2                   | tl <sub>t</sub>         | c.eq <sub>f</sub>               | 11 0010 | 50      | 32           | 2               | 114     | 72           | r               |
| pref                   | tl <sub>tu</sub>        | c.ueq <sub>f</sub>              | 11 0011 | 51      | 33           | 3               | 115     | 73           | s               |
|                        | teq                     | c.o <sub>lt</sub> <sub>f</sub>  | 11 0100 | 52      | 34           | 4               | 116     | 74           | t               |
| ldc1                   |                         | c.o <sub>lt</sub> <sub>f</sub>  | 11 0101 | 53      | 35           | 5               | 117     | 75           | u               |
| ldc2                   | tne                     | c.o <sub>le</sub> <sub>f</sub>  | 11 0110 | 54      | 36           | 6               | 118     | 76           | v               |
|                        |                         | c.o <sub>le</sub> <sub>f</sub>  | 11 0111 | 55      | 37           | 7               | 119     | 77           | w               |
| sc                     |                         | c.s <sub>f</sub>                | 11 1000 | 56      | 38           | 8               | 120     | 78           | x               |
| swc1                   |                         | c.ng <sub>le</sub> <sub>f</sub> | 11 1001 | 57      | 39           | 9               | 121     | 79           | y               |
| swc2                   |                         | c.seq <sub>f</sub>              | 11 1010 | 58      | 3a           | :               | 122     | 7a           | z               |
|                        |                         | c.ng <sub>l</sub> <sub>f</sub>  | 11 1011 | 59      | 3b           | ;               | 123     | 7b           | {               |
|                        |                         | c.l <sub>t</sub> <sub>f</sub>   | 11 1100 | 60      | 3c           | <               | 124     | 7c           | }               |
| sdcl                   |                         | c.nge <sub>f</sub>              | 11 1101 | 61      | 3d           | =               | 125     | 7d           |                 |
| sdcl                   |                         | c.le <sub>f</sub>               | 11 1110 | 62      | 3e           | >               | 126     | 7e           | ~               |
| sdcl                   |                         | c.ngt <sub>f</sub>              | 11 1111 | 63      | 3f           | ?               | 127     | 7f           | DEL             |

(1) opcode(31:26) = 0

(2) opcode(31:26) = 17<sub>ten</sub> (11<sub>hex</sub>); if fmt(25:21) = 16<sub>ten</sub> (10<sub>hex</sub>) f = s (single);  
if fmt(25:21) = 17<sub>ten</sub> (11<sub>hex</sub>) f = d (double)

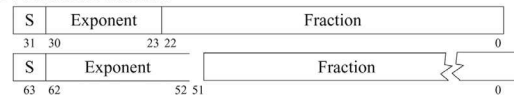
© 2014 by Elsevier, Inc. All rights reserved. From Patterson and Hennessy, *Computer Organization and Design*, 5th ed.

### IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127,  
Double Precision Bias = 1023.

### IEEE Single Precision and Double Precision Formats:

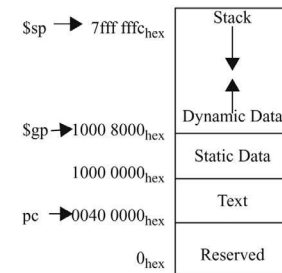


### IEEE 754 Symbols

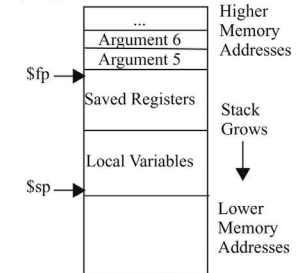
| Exponent     | Fraction | Object         |
|--------------|----------|----------------|
| 0            | 0        | ± 0            |
| 0            | ≠ 0      | ± Denorm.      |
| 1 to MAX - 1 | anything | ± Fl. Pt. Num. |
| MAX          | 0        | ±∞             |
| MAX          | ≠ 0      | NaN            |

S.P. MAX = 255, D.P. MAX = 2047

### MEMORY ALLOCATION



### STACK FRAME

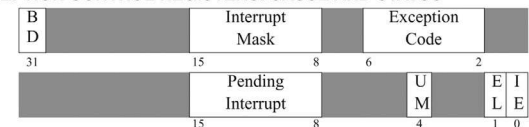


### DATA ALIGNMENT

| Double Word |      |          |      |          |      |          |      |
|-------------|------|----------|------|----------|------|----------|------|
| Word        |      |          |      | Word     |      |          |      |
| Halfword    |      | Halfword |      | Halfword |      | Halfword |      |
| Byte        | Byte | Byte     | Byte | Byte     | Byte | Byte     | Byte |
| 0           | 1    | 2        | 3    | 4        | 5    | 6        | 7    |

Value of three least significant bits of byte address (Big Endian)

### EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

### EXCEPTION CODES

| Number | Name | Cause of Exception                                  | Number | Name | Cause of Exception             |
|--------|------|---|--------|------|--------------------------------|
| 0      | Int  | Interrupt (hardware)                                | 9      | Bp   | Breakpoint Exception           |
| 4      | AdEL | Address Error Exception (load or instruction fetch) | 10     | RI   | Reserved Instruction Exception |
| 5      | AdES | Address Error Exception (store)                     | 11     | CpU  | Coprocessor Unimplemented      |
| 6      | IBE  | Bus Error on Instruction Fetch                      | 12     | Ov   | Arithmetic Overflow Exception  |
| 7      | DBE  | Bus Error on Load or Store                          | 13     | Tr   | Trap                           |
| 8      | Sys  | Syscall Exception                                   | 15     | FPE  | Floating Point Exception       |

### SIZE PREFIXES

|                  | PREFIX | SYMBOL | SIZE            | PREFIX | SYMBOL | SIZE             | PREFIX | SYMBOL | SIZE            | PREFIX | SYMBOL |
|------------------|--------|--------|-----------------|--------|--------|------------------|--------|--------|-----------------|--------|--------|
| 10 <sup>3</sup>  | Kilo-  | K      | 2 <sup>10</sup> | Kibi-  | Ki     | 10 <sup>10</sup> | Peta-  | P      | 2 <sup>40</sup> | Pebi-  | Pi     |
| 10 <sup>6</sup>  | Mega-  | M      | 2 <sup>20</sup> | Mebi-  | Mi     | 10 <sup>18</sup> | Exa-   | E      | 2 <sup>60</sup> | Exbi-  | Ei     |
| 10 <sup>9</sup>  | Giga-  | G      | 2 <sup>30</sup> | Gibi-  | Gi     | 10 <sup>21</sup> | Zetta- | Z      | 2 <sup>70</sup> | Zebi-  | Zi     |
| 10 <sup>12</sup> | Tera-  | T      | 2 <sup>40</sup> | Tebi-  | Ti     | 10 <sup>24</sup> | Yotta- | Y      | 2 <sup>80</sup> | Yobi-  | Yi     |

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together