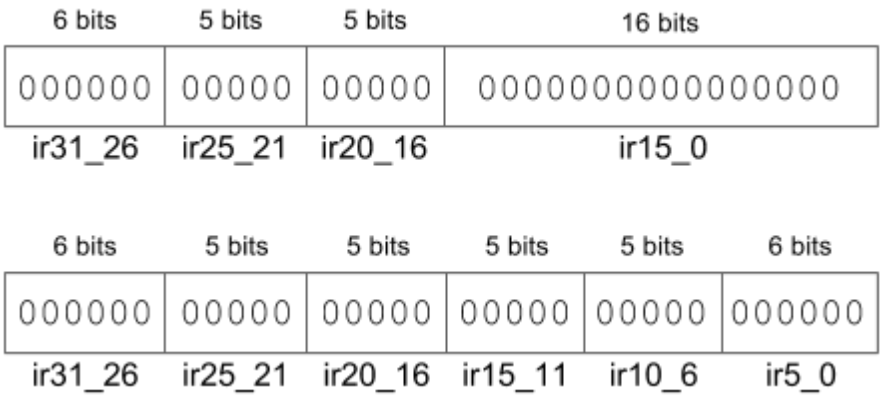
7.4 Base MIPSzy (lw, sw, addi, add): Behavior

MIPSzy instruction field bits

When designing a digital circuit to implement the MIPSzy processor, for ease of reference, bits forming an instruction field name, like ir31_26 referring to bits 31, 30, 29, 28, 27, and 26.

Figure 7.4.1: Convenient names for bit groups in a machine instruction, corresponding to fields in different instruction types.



PARTICIPATION ACTIVITY 7.4.1: Instruction field bits.

Consider the figure above.

1) Leftmost 6 instruction bits.

- ir31_26
- ir31_25
- ir5_0

2) Rightmost 6 instruction bits.
   - ir5_0
   - ir15_0

3) The rightmost 16 bits are named ir15_0, but those do not include the rightmost 6 bits, which already have a name.
   - True
   - False

Base MIPSzy behavior

For simplicity, this section defines the behavior of a base MIPSzy version. The ***base MIPSzy*** version only implements the lw add instructions. Other MIPSzy instructions will later be added to the base version.

A ***processor*** is a circuit that executes instructions. Designers commonly first described a processor's desired behavior, and circuit to implement that behavior. Such behavior can be described using a C-like language, as below (that C-like descriptio processor should not be confused with the program that the processor will execute).

The behavioral description declares key storage components as variables: instruction memory (IM), data memory (DM), reg and PC. The PC holds the address of the current instruction to execute.

PARTICIPATION ACTIVITY 7.4.2: Base MIPSzy's behavioral description.

Start ☐ 2x speed

Variables:
```
PC(32), RF(32)(32), IM(1024)(32), DM(1024)(32),
ir(32), dm_a(32), dm_rd(32), dm_wd(32), add1(32), add2(32)
```

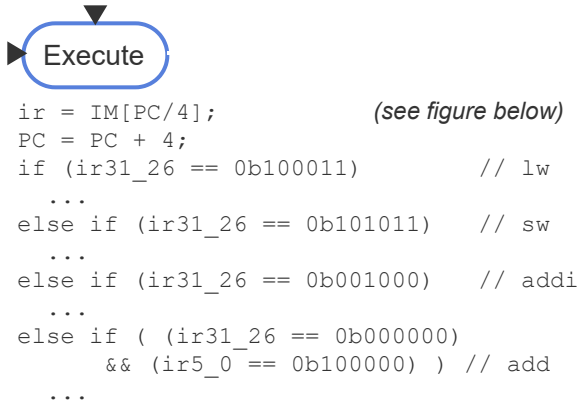


Figure 7.4.2: MIPSzy behavioral description: The Execute state's actions described using C-like code.

- ir = IM[PC/4] fetches the current instruction. PC = PC + 4 readies the PC for the next fetch.
- The if-else determines if the instruction opcode is lw, sw, addi, or add.
- Each if-else branch carries out that instruction's actions
- Variables like dm_a or add1 used as temporary values

```
ir = IM[PC/4];
PC = PC + 4;

// Assume ir31_26 etc are extracted

if      (ir31_26 == 0b100011)   { // lw
  dm_a = RF[ir25_21];
  dm_rd = DM[(dm_a-4096)/4];
  RF[ir20_16] = dm_rd;
}
else if (ir31_26 == 0b101011)   { // sw
  dm_a = RF[ir25_21];
  dm_wd = RF[ir20_16];
  DM[(dm_a-4096)/4] = dm_wd;
}
else if (ir31_26 == 0b001000)   { // addi
  add1 = RF[ir25_21];
  add2 = ir15_0;
  RF[ir20_16] = add1 + add2;
}
else if ( (ir31_26 == 0b000000)
       && (ir5_0 == 0b100000) ) { // add
  add1 = RF[ir25_21];
  add2 = RF[ir20_16];
  RF[ir15_11] = add1 + add2;
}
```

within the Execute state will become wires.

- IM and DM are 1024-word memories, 4 bytes per word, yielding 4096 items each. MIPSzy only supports word access though, so the rightmost two address bits are ignored (hence the "/4" when accessing IM or DM).
- DM is implemented as a separate memory from IM. Hence, DM addresses are first adjusted by subtracting 4096. Ex: Address 4096 becomes address 0 for DM, 5000 becomes 4, etc.. (And then /4 is performed as described above).

Table 7.4.1: Reminder: Base MIPSzy machine instructions.

Assembly	Machine
lw $t0, 0($t1)	100011 01001 01000 0000000000000000
sw $t0, 0($t1)	101011 01001 01000 0000000000000000

addi $t0, $t1, 15	001000 01001 01000 0000000000001111
add $t0, $t1, $t2	000000 01001 01010 01000 00000 100000

PARTICIPATION ACTIVITY 7.4.3: MIPSzy behavior description: The Execute state's actions.

Consider the figure above, showing the MIPSzy Execute state's actions.

1) The Execute state first reads the current instruction from IM, into a variable named ir.
   - ○ True
   - ○ False

2) The Execute state increments PC by 4 before reading the IM.
   - ○ True
   - ○ False

3) The figure shows how field names like ir31_26 are associated with ir.
   - ○ True
   - ○ False

4) The if-else statement compares ir31_26 with the opcodes for lw, sw, addi, and add.
   - ○ True
   - ○ False

5)

If ir31_26 is 001000, the Execute state will sum the values from two registers.

- ○ True
- ○ False

6) addi's actions include: `add1 = RF[ir25_21];` That statement reads RF using bits ir25_21 as the register address.

- ○ True
- ○ False

7) addi's actions use variables add1 and add2. Those variables will become registers in the processor circuit.

- ○ True
- ○ False

8) The add instruction's actions write to RF[ir15_11].

- ○ True
- ○ False

9) Because DM is implemented in a separate memory as IM, addresses intended for DM have 4096 subtracted first.

- ○ True
- ○ False

Note to instructors: Word-alignment.

For the authors, defining MIPSzy involved many tradeoffs between goals like simplicity (to assist learning) and MIPS consistency (for simulator use and smooth lead-in to a next course). Keeping addressing consistent was critical for the latter goal, but unfortunately requires the various address adjustments that appear above (like having a 1024-word memory rather than 4096, and like dividing by 4 to ignore byte addresses).

Provide feedback on this section