7.5 Base MIPSzy: Processor design

A brief note from your instructor:

This CPU can only perform a lw and sw with an offset of zero. I will post a slightly modified version that we will be using

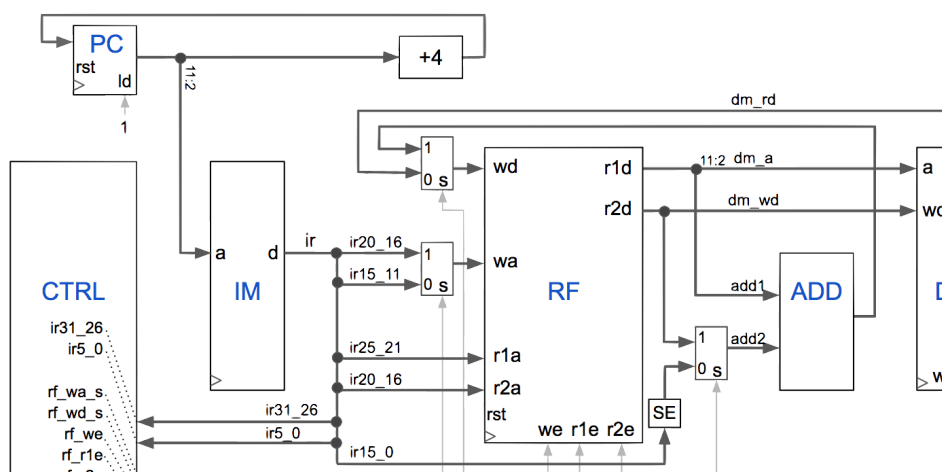
The base MIPSzy's processor design will include several components: instruction memory (IM), data memory (DM), registe (ADD), and control logic (CTRL). To implement the base MIPSzy processor, a designer can create a circuit of components t MIPSzy behavioral description's actions, and then convert the behavioral description into control logic actions, implemente combinational circuit in CTRL, that configure the other components to carry out the current instruction's actions.

PARTICIPATION ACTIVITY 7.5.1: Creating a processor circuit that supports the MIPSzy behavioral description's actions.

Start 2x speed

```
ir = IM[PC/4];
PC = PC + 4;
// Assume ir31_26... extracted
// lw
if (ir31_26 == 0b100011)  {
  dm_a   = RF[ir25_21];
  dm_rd  = DM[(dm_a-4096)/4];
  RF[ir20_16] = dm_rd;
}
// sw
else if (ir31_26 == 0b101011){
  dm_a     = RF[ir25_21];
  dm_wd    = RF[ir20_16];
  DM[(dm_a-4096)/4] = dm_wd;
}
// addi
```

```
else if (ir31_26 == 0b001000){
  add1 = RF[ir25_21];
  add2 = ir15_0;
  RF[ir20_16] = add1 + add2;
}
// add
else if ((ir31_26 == 0b000000)
      &&(ir5_0 == 0b100000)){
  add1 = RF[ir25_21];
  add2 = RF[ir20_16];
  RF[ir15_11] = add1 + add2;
}
```



In addition to key components (IM, RF, ADD, DM, PC), the processor's circuit also includes muxes wherever a component's i from two different sources (to support different instructions); the control logic will set the mux's select line to pass the corr the current instruction. Also, the processor's circuit includes a ***sign-extender*** (SE) component, which extends a two's-comp number into a wider number by prepending 1's (if the leftmost bit was 1) or 0's (if the leftmost bit was 0), in this case exten instruction's 16-bit immediate to a 32-bit input for the adder.

PARTICIPATION ACTIVITY 7.5.2: Creating a circuit to support the MIPSzy behavioral description's actions.

1) The PC holds the current instruction address. To support fetching the current instruction, the PC's output is connected to _____.
   - ○ DM's address input
   - ○ IM's address input
   - ○ RF's address input

2) The PC has _____ different sources of data input.
   - ○ 1

- 2
- 3

3) The PC register's load input is ____.
   - tied to 1
   - set to 1 or 0 by CTRL

4) The comparison of ir31_26 with 100011 is done in the _____ component.
   - ADD
   - CTRL

5) The lw instruction reads an RF register using RF port _____.
   - 1
   - 2

6) The lw instruction reads DM onto dm_rd (DM read data). That data is written into the RF register specified by what address?
   - ir25_21
   - ir20_16
   - ir15_11

7) The sw instruction reads an address from an RF register similar to lw. However, sw then reads a second RF register on RF read port ____, whose value connects to DM's data input.
   - 1

- 2

8) The addi instruction connects ir15_0 to ADD's _____ input.
   - top
   - bottom

9) MIPS supports byte addressing, but MIPSzy only supports word addresses, ignoring byte addresses. Thus, MIPSzy's design ignores the rightmost ___ bits of an address.
   - 2
   - 4

10) For simplicity, the MIPSzy design ignores the leftmost bits of memory addresses, considering only bits 11:2. As such, the design _____ IM addresses outside of 0-4092 and DM addresses outside of 4096-8188.
   - detects
   - ignores

11) MIPSzy's high-level behavior subtracted 4096 from data memory addresses due to DM being implemented in a separate memory from IM. How is such subtraction carried out in MIPSzy's design?
   - Using a subtractor
   -

Using the ADD component

- ○ No such subtraction is performed

Base MIPSzy's control logic actions

The behavioral description's actions can be replaced by control logic actions that carry out the desired high-level actions or circuit, as shown below. Any control signal not explicitly set (with 0 or 1) on a given pass through the Execute state is implic

Figure 7.5.1: Base MIPSzy behavioral description's Execute state actions, and corresponding control logic actions making use of the above processor circuit.

Behavioral description's actions:	Control logic actions:

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

```
// PC11:2 connected to IM's address sets ir
// PC's ld input tied to 1 loads PC + 4

                        // lw
if      (ir31_26 == 0b100011)  {
   rf_r1e = 1; // Reads RF[ir25_21] onto dm_a
   dm_re = 1;  // Reads DM using dm_a 11:2
   rf_wd_s = 0;  rf_wa_s = 1;  rf_we  = 1;
}
                        // sw
else if (ir31_26 == 0b101011)  {
   rf_r1e = 1;
   rf_r2e = 1;
   dm_we = 1;
}
                        // addi
else if (ir31_26 == 0b001000)  {
   rf_r1e = 1;
   add_add2_s = 0;
   rf_wd_s = 1;  rf_wa_s = 1;  rf_we = 1;
}
                         // add
else if ( (ir31_26 == 0b000000)
        && (ir5_0   == 0b100000) ){
   rf_r1e = 1;
   rf_r2e = 1;  add_add2_s = 1;
   rf_wd_s = 1;  rf_wa_s = 0;  rf_we = 1;
}
```

The control logic CTRL can be implemented using a standard combinational circuit design process, based on the control lc created above.

PARTICIPATION ACTIVITY 7.5.3: Base MIPSzy's control logic actions.

Consider the figure above showing MIPSzy's control logic actions.

1) Which causes ir = IM[PC/4]?

- ir_ld = 1;
- IM_re = 1;
- None

2) Which control logic action causes PC = PC + 4?

- none
- plus4 = 1;

3) lw: Which lw control logic action(s) carries out statement RF[ir20_16] = dm_rd?

- rf_r1e = 1;
- dm_re = 1;
- rf_wd_s = 0;
  rf_wa_s = 1;
  rf_we = 1;

4) sw: Which sw control logic action carries out statement DM[(dm_a-4096)/4] = dm_wd?

- rf_r1e = 1;
- rf_r2e = 1;
- dm_we = 1;

5) addi: Which addi control logic action(s) carries out statement add1 = RF[ir25_21]?

- rf_r1e = 1;
- add_add2_s = 0;
- rf_wd_s = 1;
  rf_wa_s = 0;
  rf_we = 1;

6) add: For the add instruction, high-level behavior action add2 = RF[ir20_16] becomes the control actions:
rf_r2e = 1; add_add2_s = 1;
Why is add_add2_s set to 1?

- To pass r2d to the adder
- To pass ir15_0 to the adder
- To enable the adder

7) Every written control signal in the above MIPSzy control logic actions must be an output of the CTRL component.

- True
- False

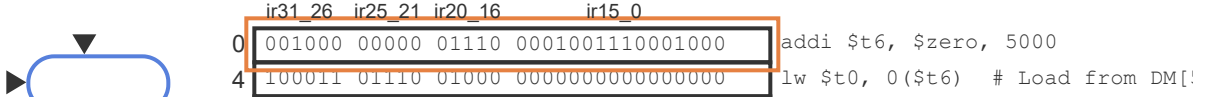
Executing instructions on the base MIPSzy processor design

The base MIPSzy processor consists of the key components of CTRL, PC, IM, RF, ADD, and DM, as below.

A program would be pre-loaded into IM. Upon starting the processor, power-on reset circuitry (not shown) would set the rst PC and RF to 1 for a short duration, clearing those components' storage elements to 0's. PC's 0 reads the instruction at IM[opcode flows to the control logic CTRL, which sets appropriate control outputs to carry out the instruction (namely, having values waiting at RF's wd, wa, and we inputs, which take effect on the next rising clock).

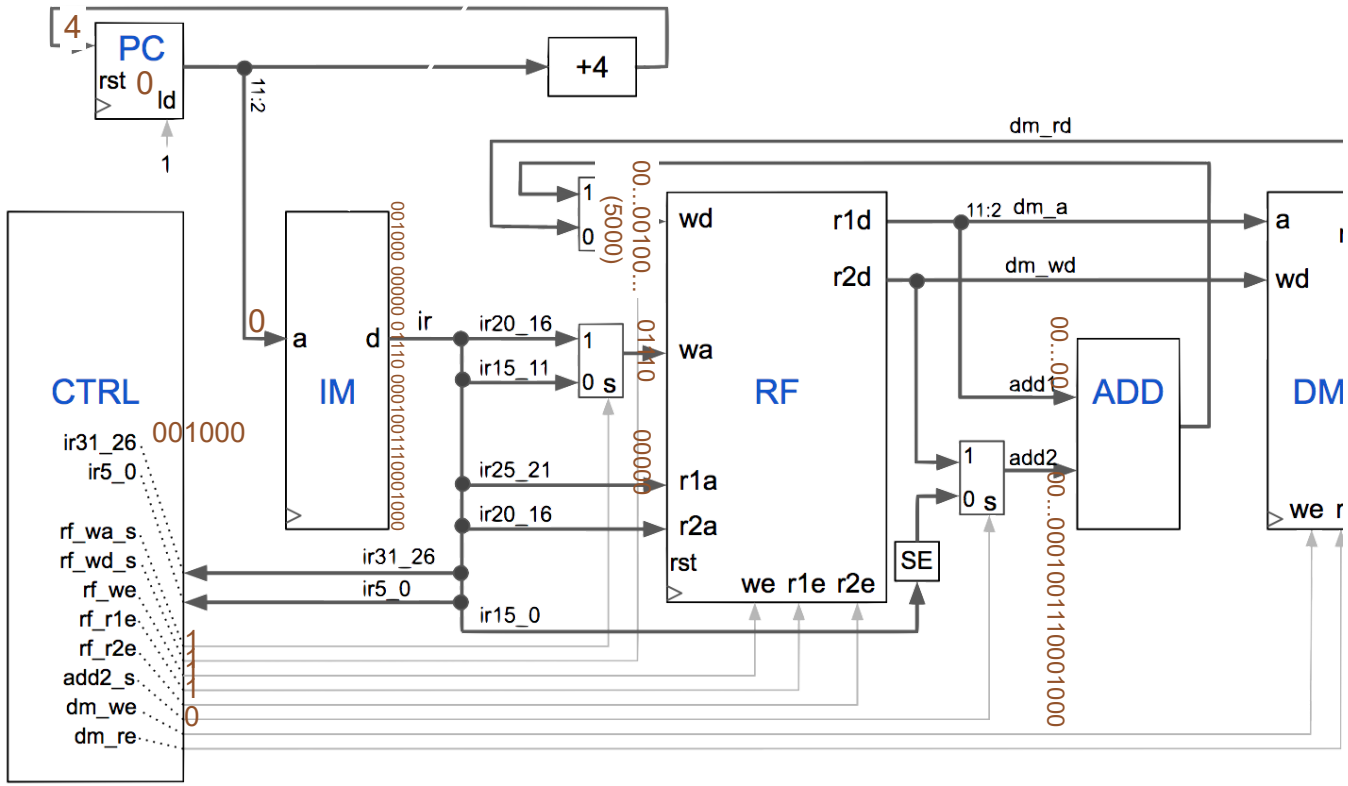
PARTICIPATION ACTIVITY 7.5.4: Executing an instruction on the MIPSzy processor: At the end of the clock cycle for addi, the result 5000 is ready to be written into RF's $t6 (01110) on the next cycle.

Start 2x speed



```
Execute
//addi
rf_r1e = 1;
add_add2_s = 0;
rf_wd_s = 1;  rf_wa_s = 1;  rf_we = 1;
```

```
8  000000 01000 01000 01001 00000 100000  add $t1, $t0, $t0 # Double the
12 101011 01110 01001 0000000000000000    sw $t1, 0($t6)   # Store to DM[5
```



One should examine the Execute state's control logic actions for each fetched instruction's opcode (recalling that all other c are set with 0), and consider how those actions cause items to flow through the components.

PARTICIPATION ACTIVITY 7.5.5: Executing addi on the MIPSzy processor.

Consider the MIPSzy processor above, carrying out the addi instruction at IM[0]. Assume addi's opcode of 001000 has already flowed back to CTRL.

1) CTRL sets output rf_r1e to ___.
   - ○ 0
   - ○ 1

2) The wires labeled add2 have a mux. CTRL sets that mux's select input to ___.
   - ○ 0
   - ○ 1

3) ADD will add the read register and ir15_0. The result should flow back to RF's write data input wd. Is a mux involved?
   - ○ Yes
   - ○ No

4) ADD will add the read register and ir15_0. The result should be written to RF[ir20_16]. Is a mux involved?
   - ○ Yes
   - ○ No

5) Is DM either written or read by addi's actions?
   - ○ Yes
   - ○ No

6) Is ir5_0 used by the control logic for

addi?

- Yes
- No

CHALLENGE ACTIVITY 7.5.1: Instruction execution on MIPSzy processor.

Start

Enter the bits for each item.

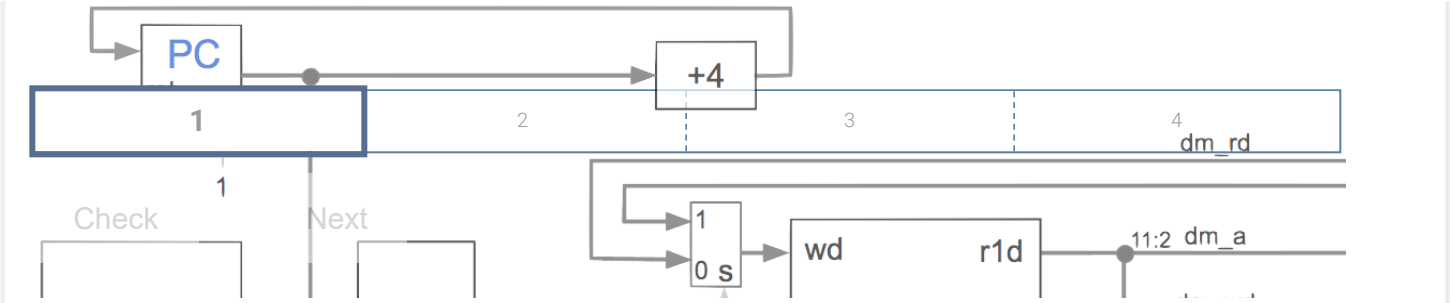
asm: lw $t1, 0($t7)

ir: 100011 01111 01001 000000(
$t7 $t1

Ex: 110010

Ex: 01000

Ex: 01000



Provide feedback on this section