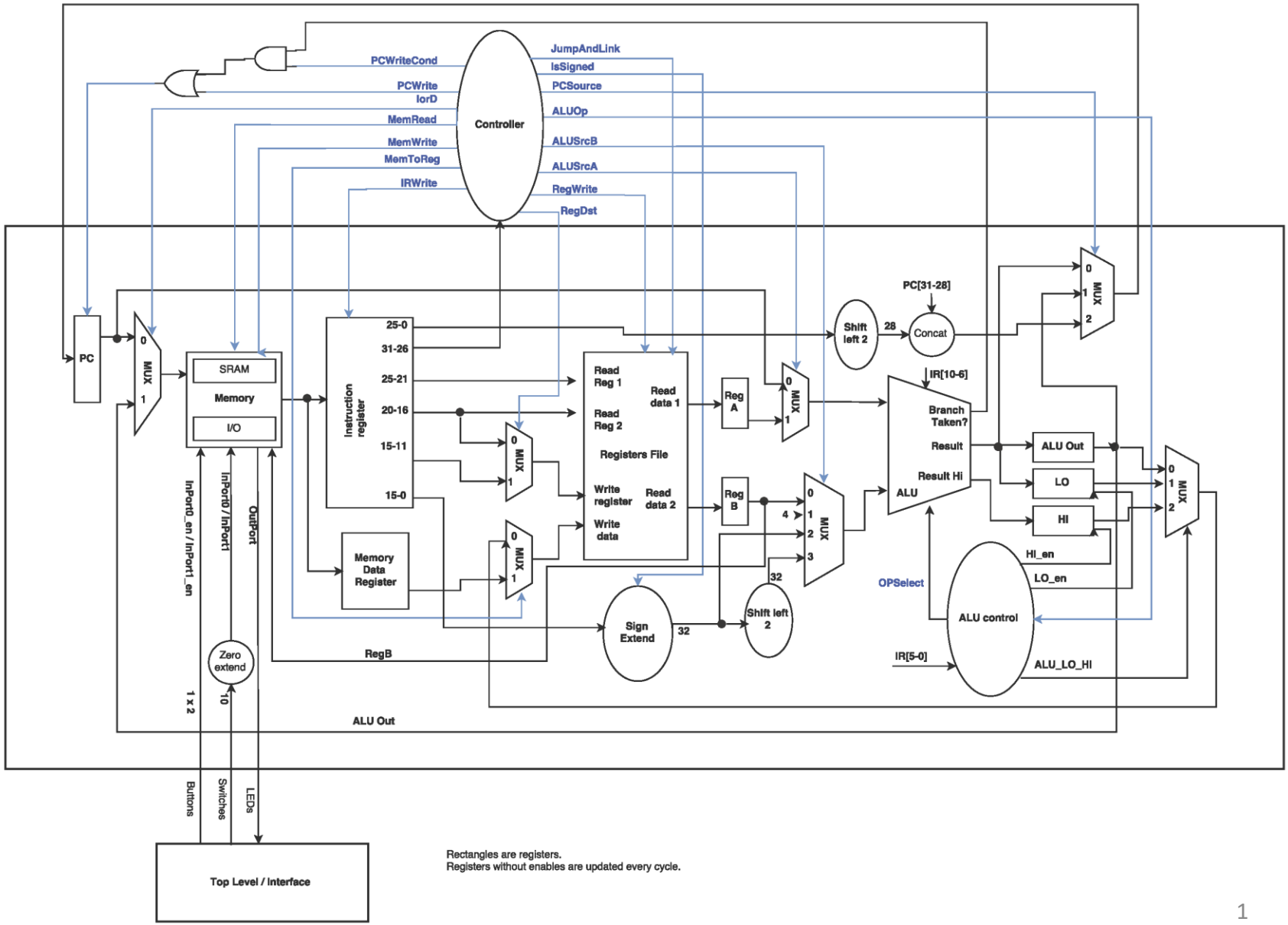
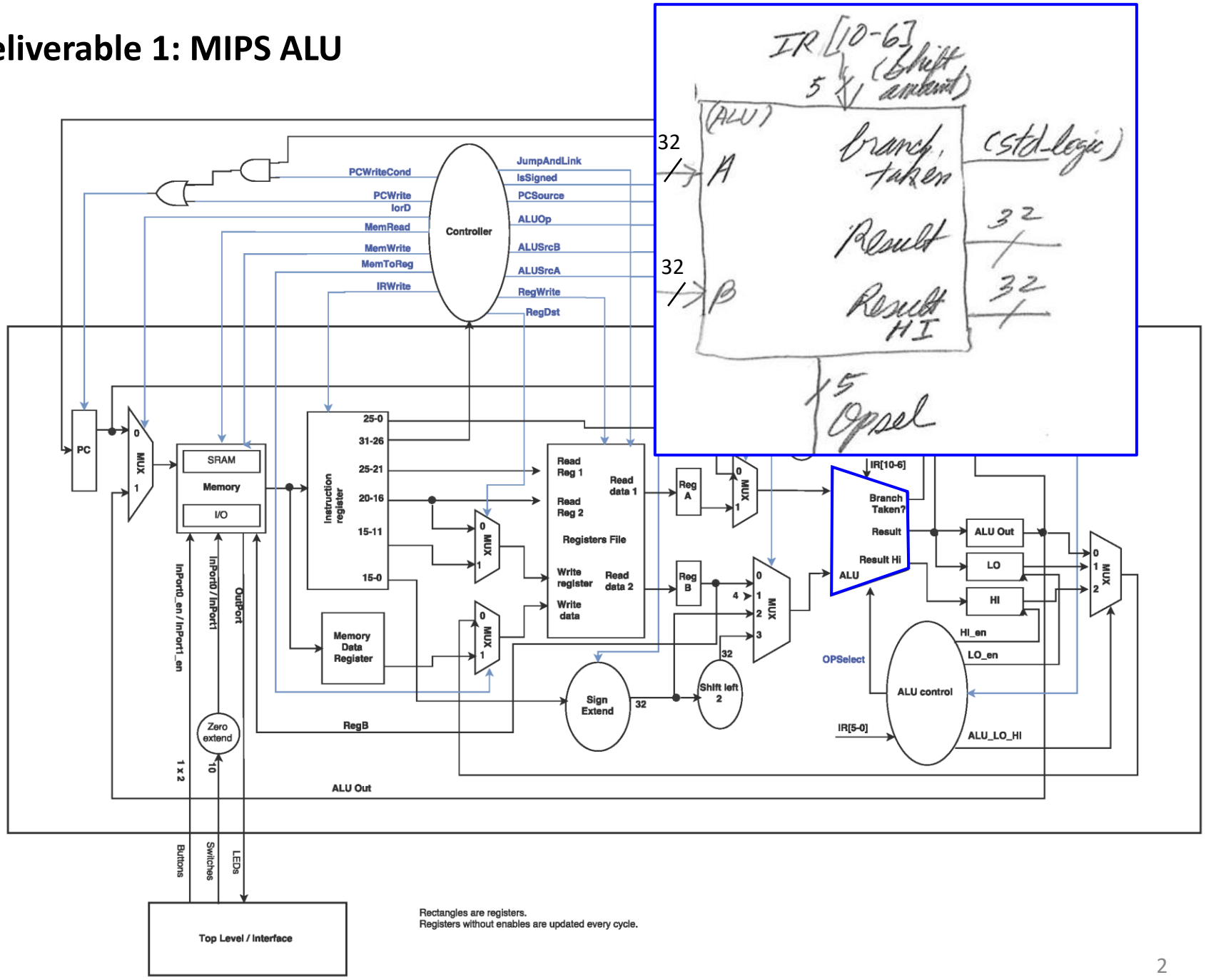
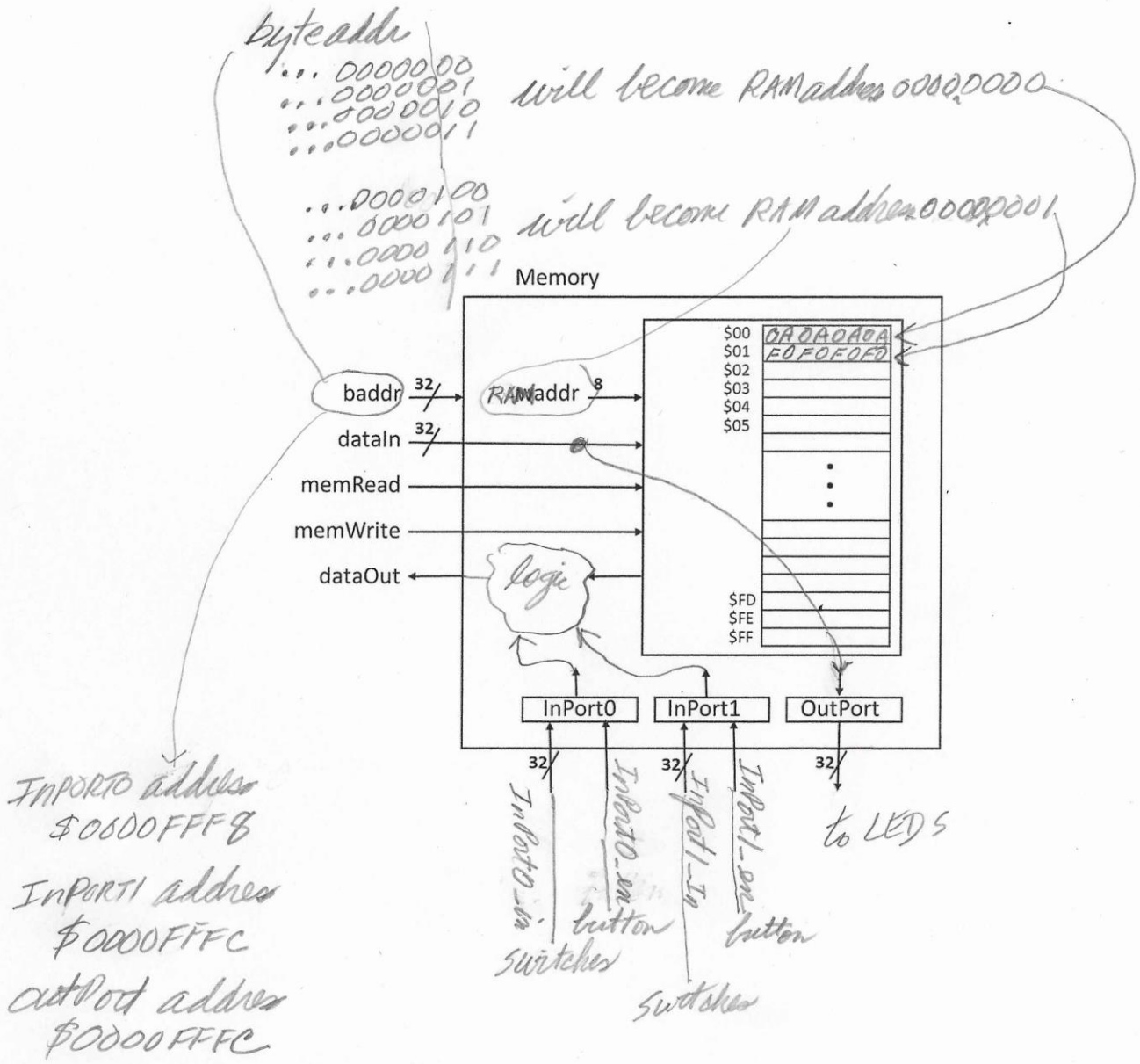
MIPS-4712 General Architecture



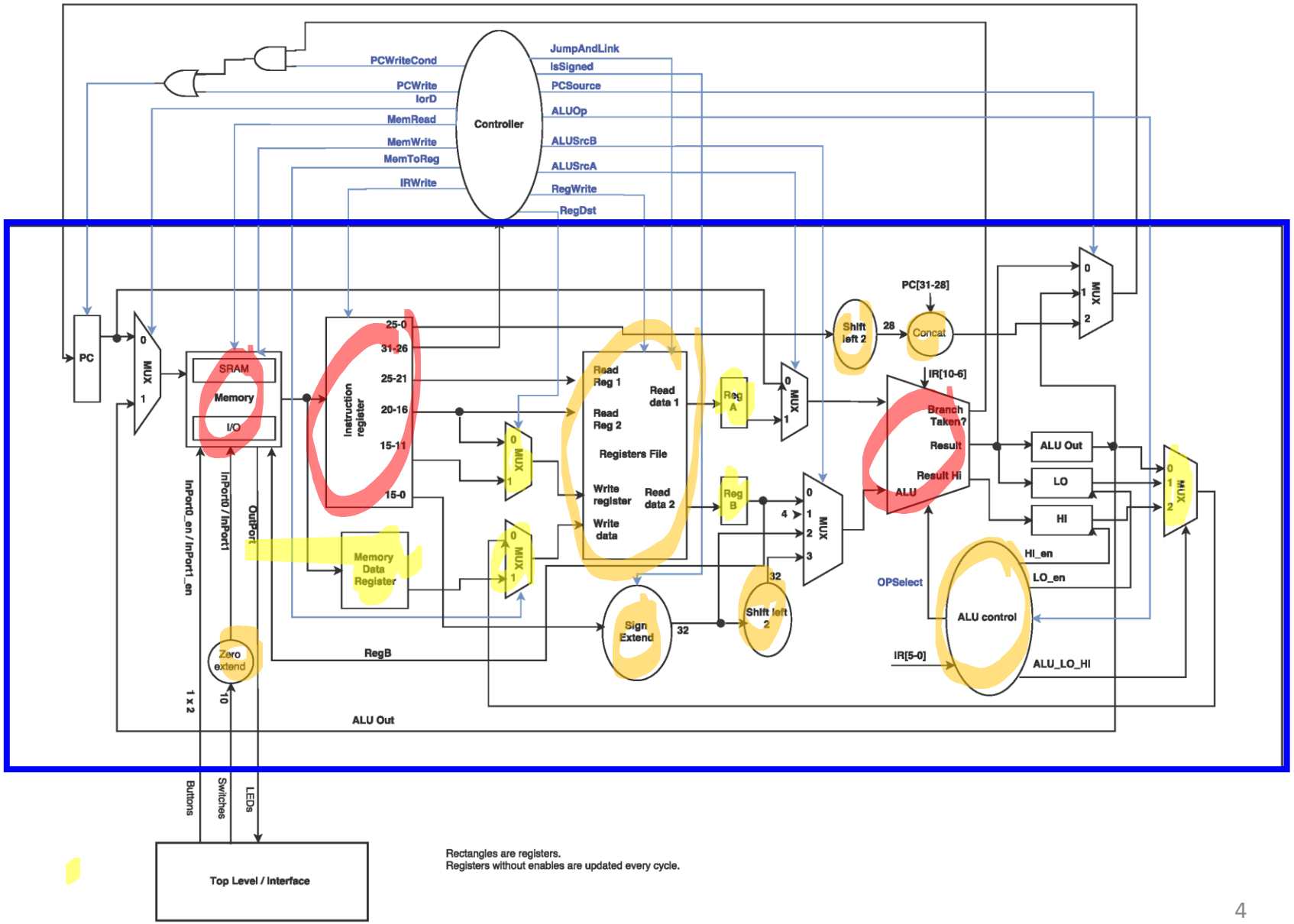
Deliverable 1: MIPS ALU



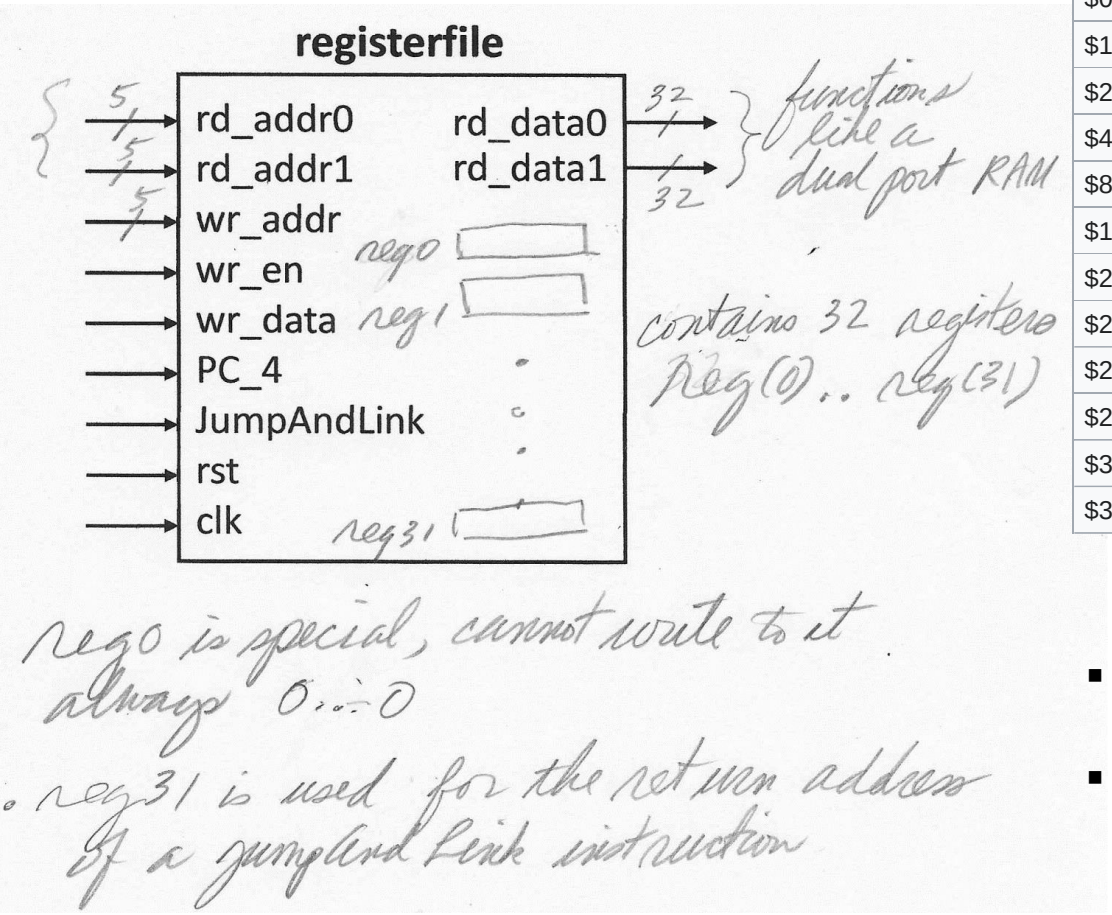
Deliverable 2: MIPS-4712 Memory/Port Module



Deliverable 3: Datapath



MIPS-4712 Register File



Number	Name	Comments
$0	$zero, $r0	Always zero
$1	$at	Reserved for assembler
$2, $3	$v0, $v1	First and second return values, respectively
$4, ..., $7	$a0, ..., $a3	First four arguments to functions
$8, ..., $15	$t0, ..., $t7	Temporary registers
$16, ..., $23	$s0, ..., $s7	Saved registers
$24, $25	$t8, $t9	More temporary registers
$26, $27	$k0, $k1	Reserved for kernel (operating system)
$28	$gp	Global pointer
$29	$sp	Stack pointer
$30	$fp	Frame pointer
$31	$ra	Return address

- **Name** is the symbolic name used in the MIP assembly language instruction.
- **Number** is the actual register number.

Example:

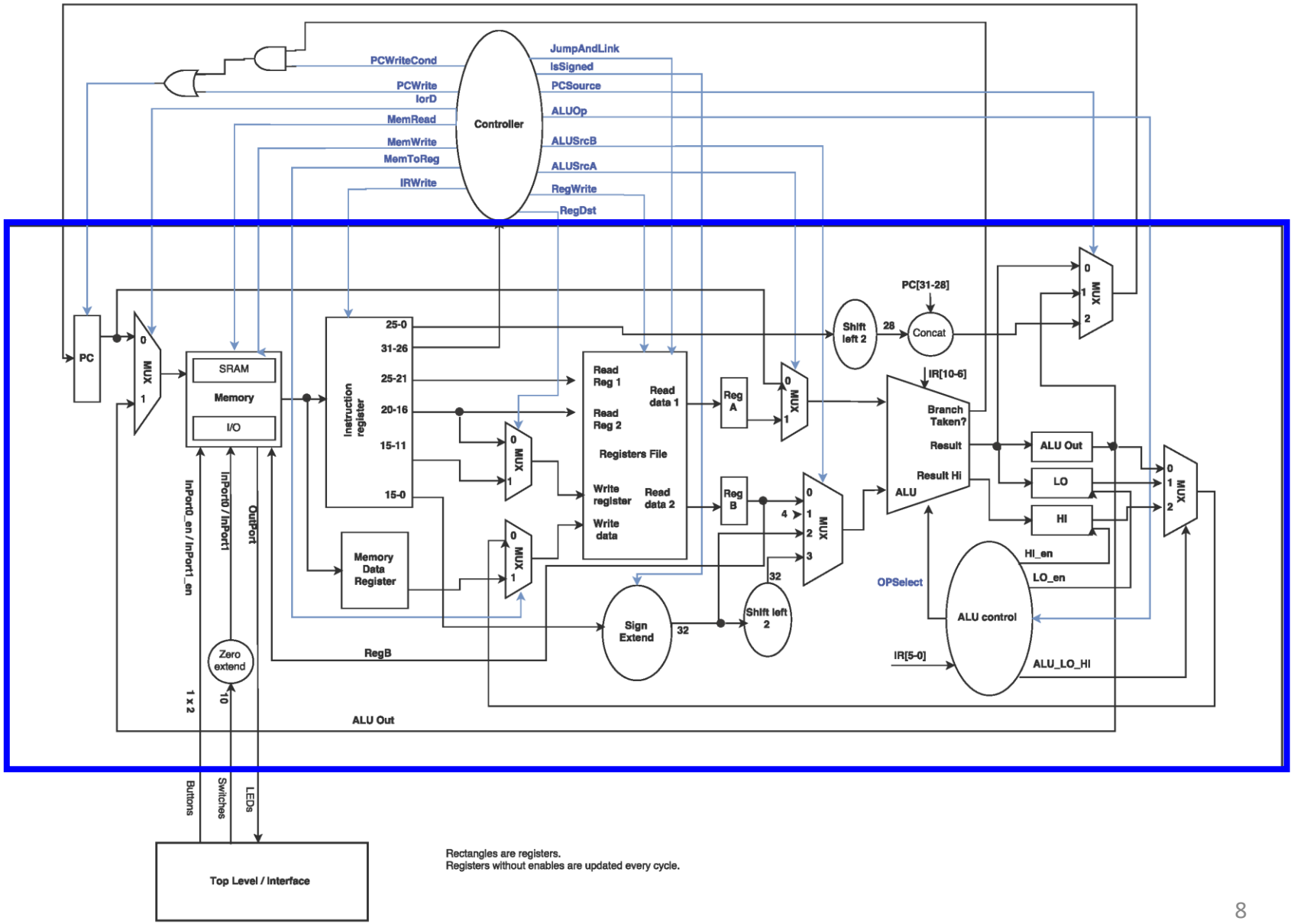
addu $s3, $s1, $s2

means (reg19) = (reg17) + (reg18)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity registerfile is
   port(
      clk : in std_logic;
      rst : in std_logic;

      rd_addr0 : in std_logic_vector(4 downto 0); --read reg 1
      rd_addr1 : in std_logic_vector(4 downto 0); --read reg 2

      wr_addr : in std_logic_vector(4 downto 0); --write register
      wr_en : in std_logic;
      wr_data : in std_logic_vector(31 downto 0); --write data

      rd_data0 : out std_logic_vector(31 downto 0); --read data 1
      rd_data1 : out std_logic_vector(31 downto 0); --read data 2
            --JAL
            PC_4 : in std_logic_vector(31 downto 0);
            JumpAndLink : in std_logic
      );
end registerfile;
```

```
architecture sync_read of registerfile is
   type reg_array is array(0 to 31) of std_logic_vector(31 downto 0);
   signal regs : reg_array;
begin
   process (clk, rst) is
   begin
      if (rst = '1') then
         for i in regs'range loop
            regs(i) <= (others => '0');
         end loop;
      elsif (rising_edge(clk)) then
         if (wr_en = '1') then
            regs(to_integer(unsigned(wr_addr))) <= wr_data;
                                    regs(0) <= (others => '0');
         end if;
         if(JumpAndLink = '1') then
             regs(31) <= PC_4;
         end if;
         rd_data0 <= regs(to_integer(unsigned(rd_addr0)));
         rd_data1 <= regs(to_integer(unsigned(rd_addr1)));
      end if;
   end process;
end sync_read;
```

Deliverable 3: Datapath



Deliverable 3: RTL View of Datapath

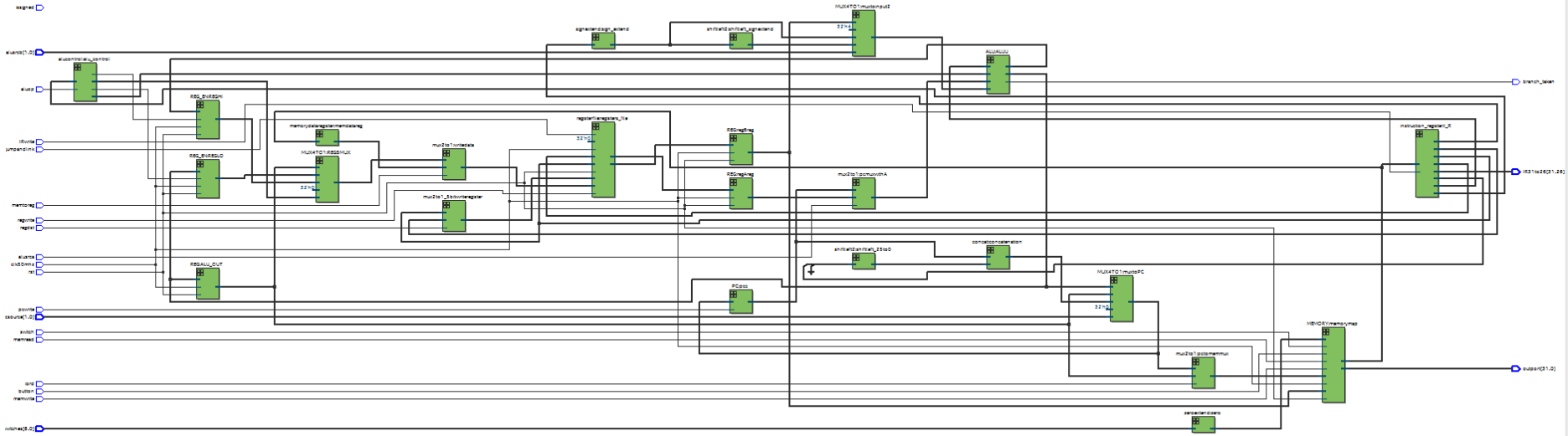
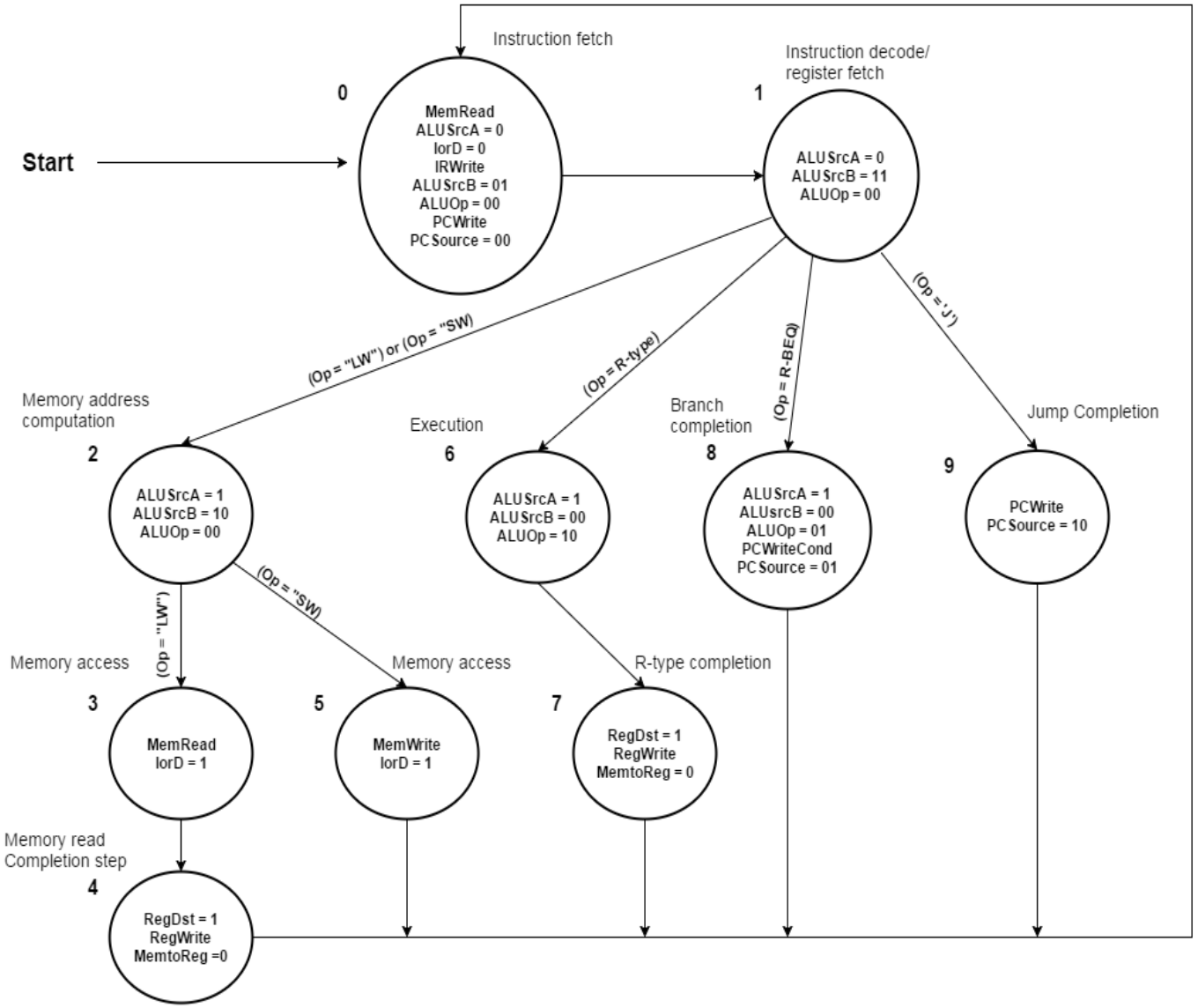


Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed



General "ASM Chart" for Controller

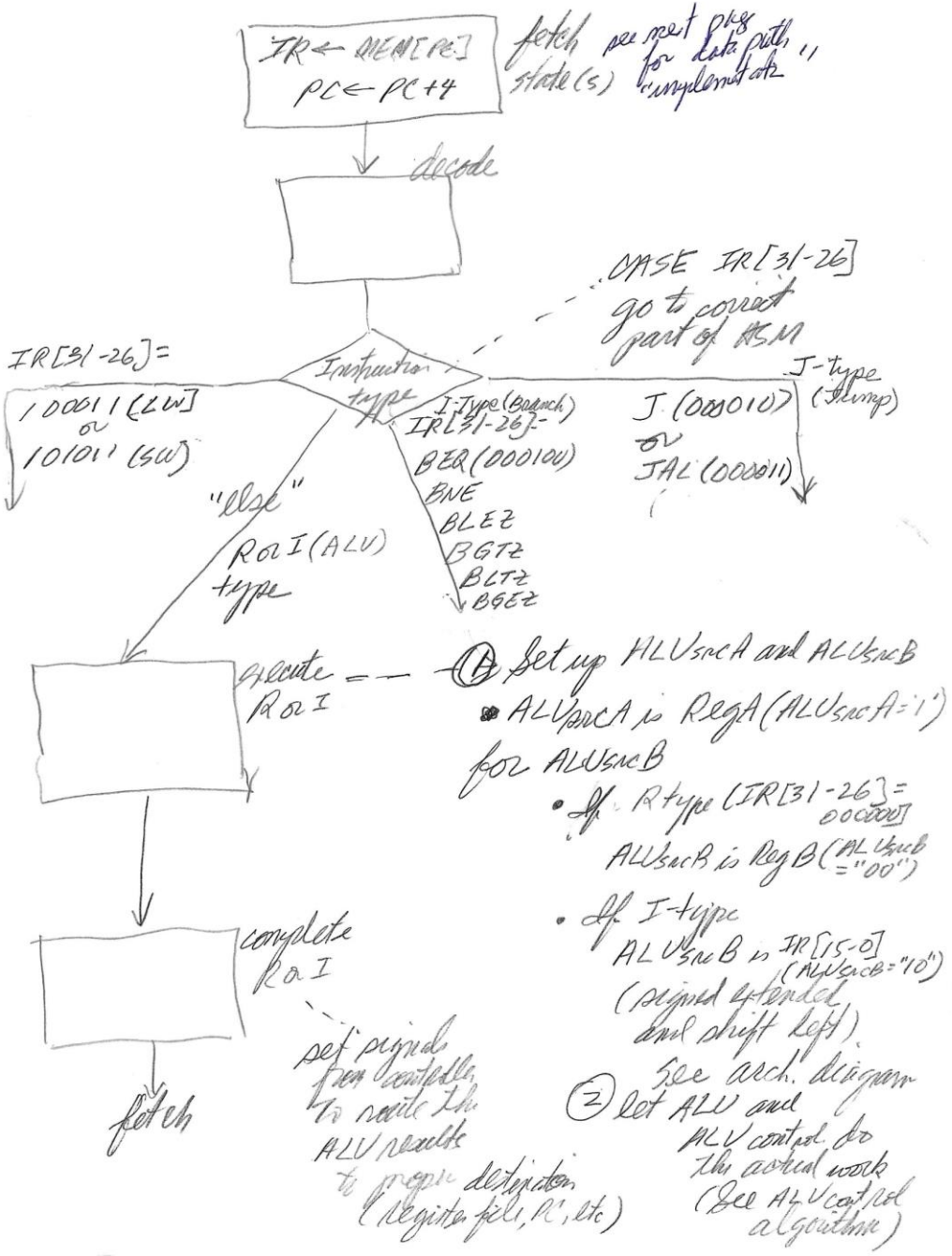


Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed

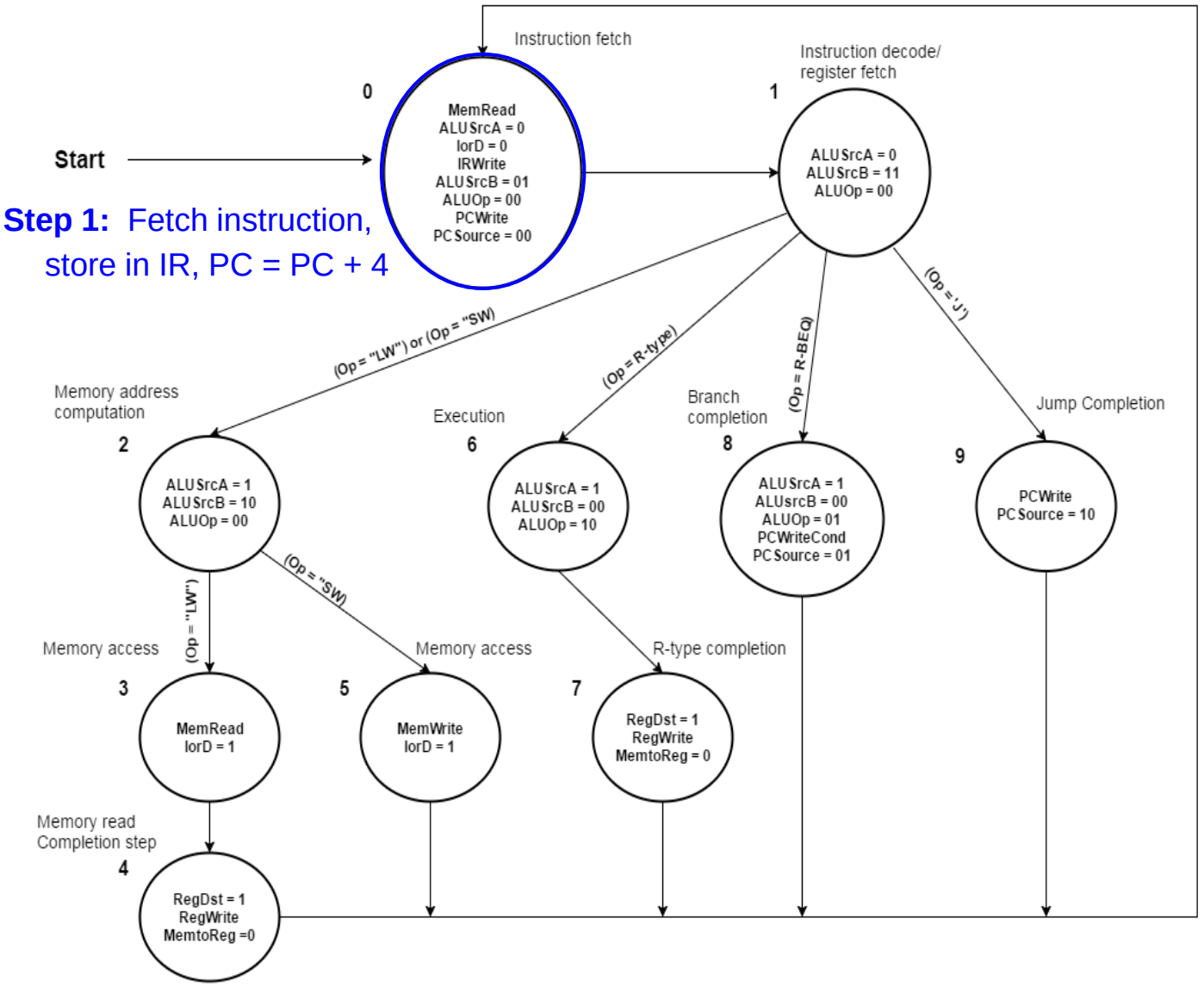
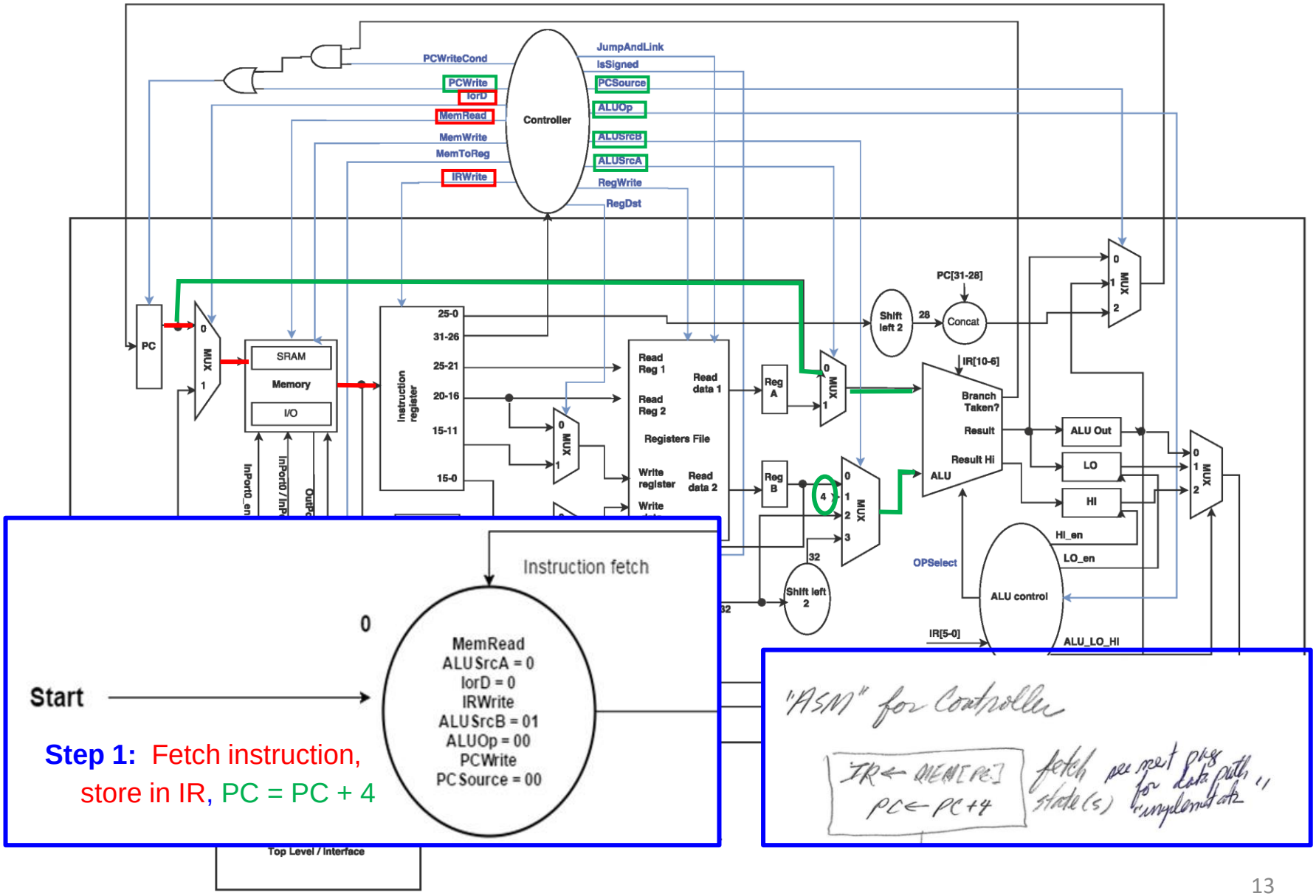


Illustration of the Instruction Fetch State



R-Type Instruction Format

31-26	25-21	20-16	15-11	10-6	5-0
opcode (000000)	*rs*	*rt*	*rd*	*shamt*	*funct*

- ***rs*** and ***rt*** are operand registers; ***rd*** is the destination register.
- **R-Type:** Register-based ALU operations.
- ***opcode*** is always 000000.
- The actual instruction code is in the ***funct*** field (5-0).
- ***shamt*** is the "shift amount" for shift instructions.
- MIPS-4712 R-type instructions: (See MIPS instruction Set Manual for details):
  add, addu, and, mfhi, mflo, mult, multu, or, sll, sltu, sra, srl, sub, subu, xor
- Example: and $s4, $s2, $s3 (s4 = s2 and s3) or (reg20 = reg18 and reg19)
  000000 10010 10011 10100 00000 100100

I-Type Instruction Format (ALU immediate)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*immed.*

- ***rs*** is operand A register and ***rt*** is destination register.
- ***immed.*** is operand B.
- MIPS-4712 I-type ALU insts.:addiu, andi, ori, "subiu" (does not exist), xori
- Example: addiu $s2, $s1, 7 (s2 = s1 + 7) or (reg18 = reg17 + 7)
  001100 10011 01010 0000000000000111

I-Type Instruction Format (Load or Store)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

- ***rs***: address base register; ***rt***: source (store) or destination (load) location in register file; ***address***: offset address
- MIPS-4712 I-type load/store instructions: lw, sw
- Example: lw $s2, 0xA($Zero) 100011 00000 10010 0000000000101000

I-Type Instruction Format (Branch)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

- ***rs***: operand A for conditional branch; ***rt***: operand B for conditional branch; ***address***: offset address for branch (+ or – locations relative to next inst.)
- MIPS-4712 I-type branch instructions: beq, bne, jr
- Example: beq $s2 , $zero, 8 000100 10010 00000 0000000000000011 (assumption: This beq instruction in in Location 4.)

J-Type Instruction Format (Jump)

31-26	25-0
opcode	*target*

- ***target***: destination location for jump operation
- MIPS-4712 J-type instructions: j, jal
- Example: j8 000010 00000000000000000000001000

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed

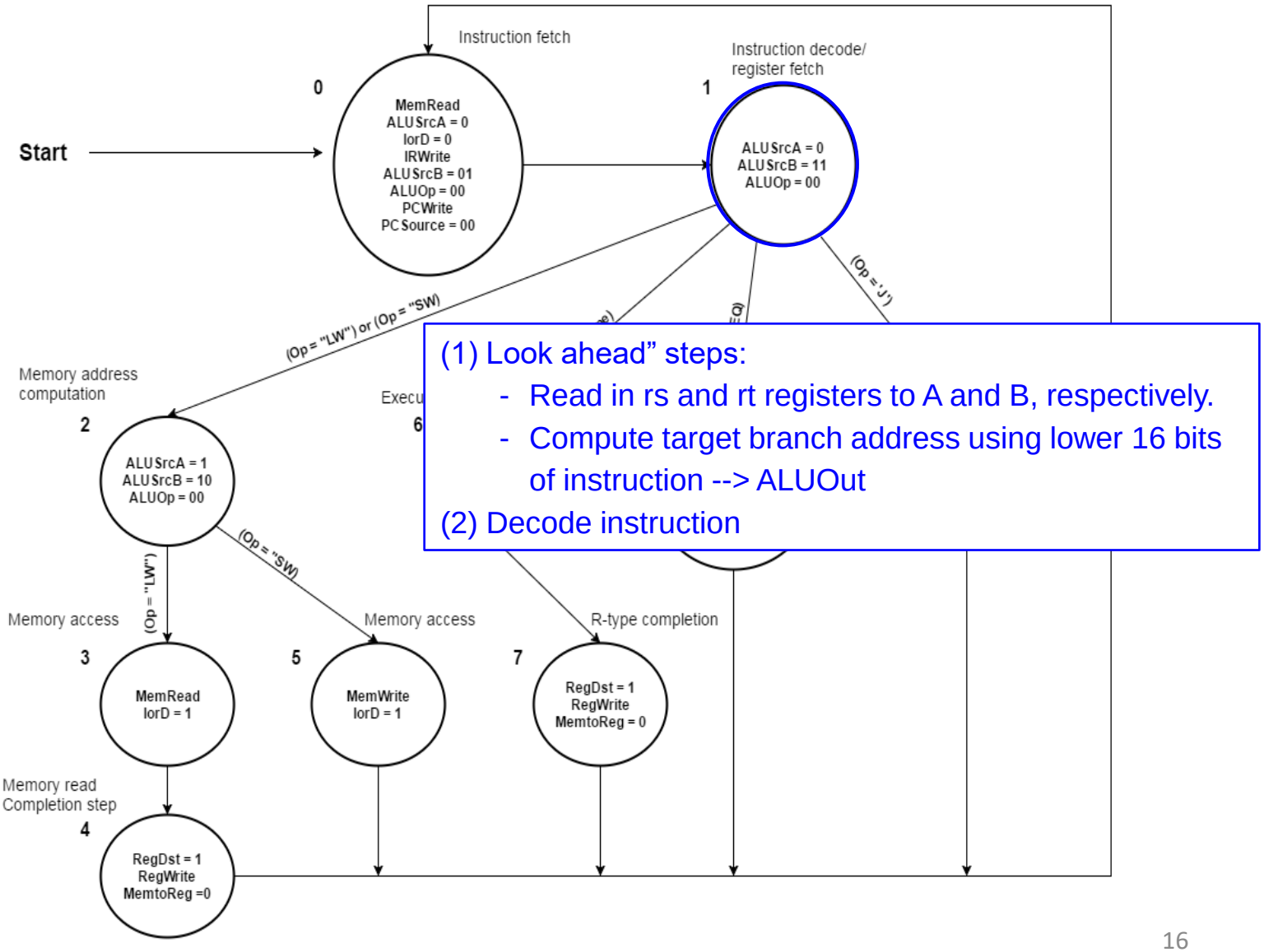
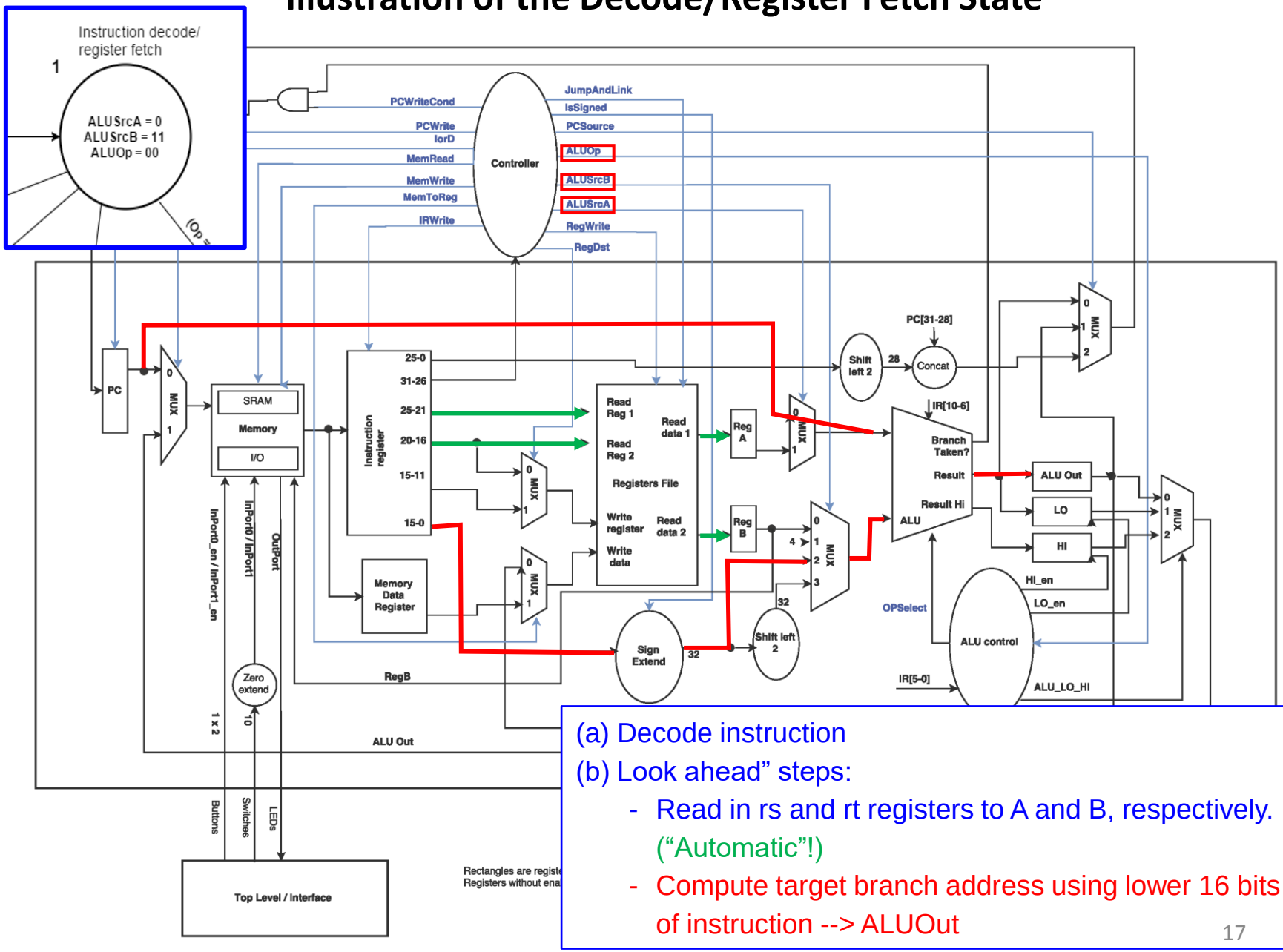


Illustration of the Decode/Register Fetch State



(a) Decode instruction

(b) Look ahead" steps:

- Read in rs and rt registers to A and B, respectively. ("Automatic"!)
- Compute target branch address using lower 16 bits of instruction --> ALUOut

R-Type Instruction Format

31-26	25-21	20-16	15-11	10-6	5-0
opcode (000000)	*rs*	*rt*	*rd*	*shamt*	*funct*

I-Type Instruction Format (ALU immediate)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*immed.*

I-Type Instruction Format (Load or Store)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

I-Type Instruction Format (Branch)

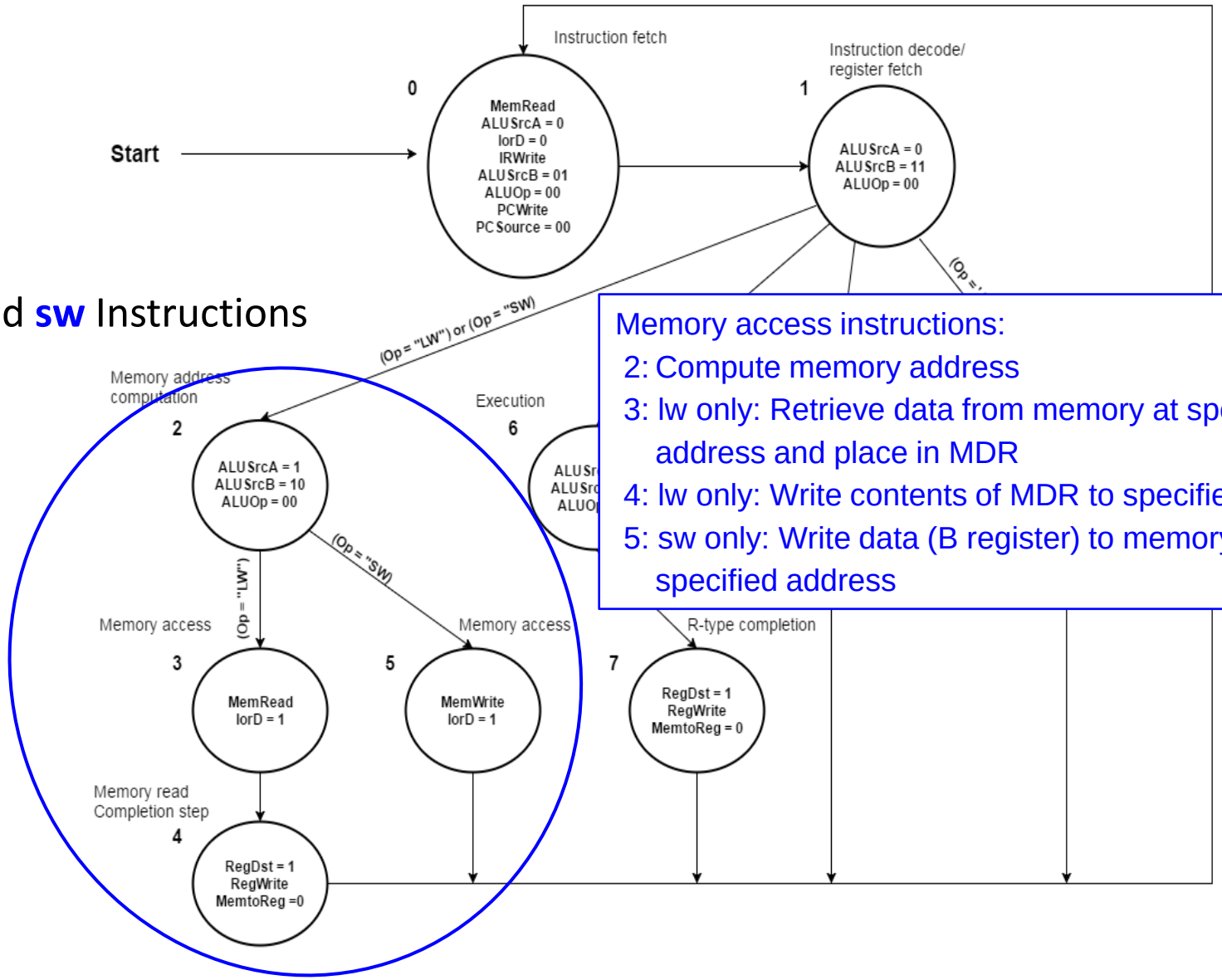
31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

J-Type Instruction Format (Jump)

31-26	25-0
opcode	*target*

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed

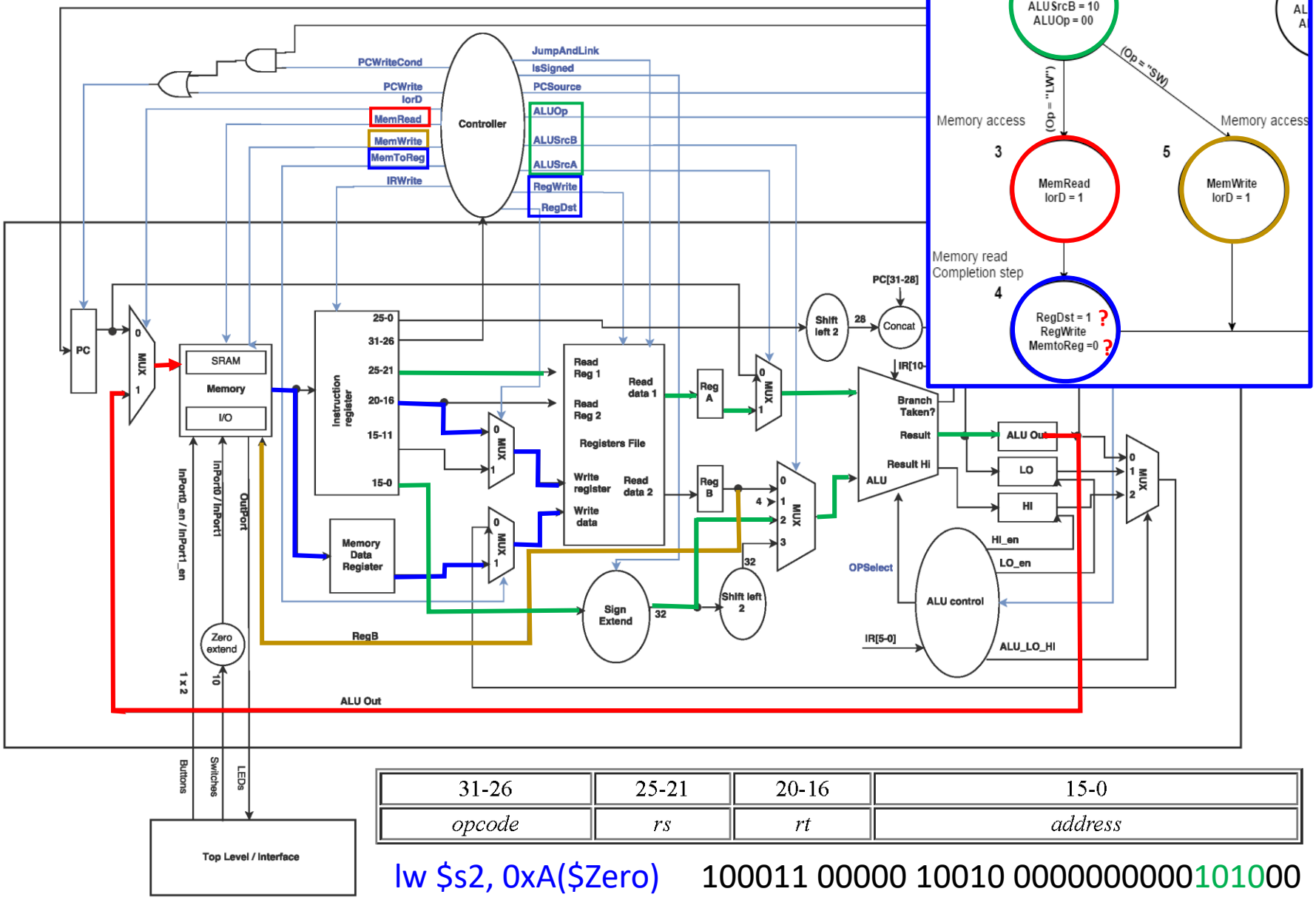


lw and **sw** Instructions

Memory access instructions:

- 2: Compute memory address
- 3: lw only: Retrieve data from memory at specified address and place in MDR
- 4: lw only: Write contents of MDR to specified register
- 5: sw only: Write data (B register) to memory at specified address

Illustration of the lw and sw Instructions



31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

lw $s2, 0xA($Zero) 100011 00000 10010 0000000000101000

s2 <= mem (A + 0);

TestCase1.mif

```
% Program RAM Data %        -- This program will test these instructions :
                            -- lw, addu, and, xor, or, sub, multu, and j
00 : 10001100000100010000000000100100;          -- lw $s1, 0x9($Zero)
     /     load word in adress 0x9 + zero to s1     // s1/r17 = 4
01 : 10001100000100100000000000101000;          -- lw $s2, 0xA($zero)
     /     load word in adress 0xA  // s2/r18 = 5
02 : 00000010001100101001100000100001;          -- addu $s3, $s1, $s2
     /     s3 = s1 + s2                          // s3/r19 = 9
03 : 00000010010100111010000000100100;          -- and $s4, $s2, $s3
     /     s4 = s2 and s3                        // s4/r20 = 1
04 : 00000010011101001010100000100110;          -- xor $s5, $s3, $s4
     /     s5 = s3 xor s4                        // s5/r21 = 8
05 : 00000010011100011011000000100101;          -- or  $s6, $s3, $s1
     /     s6 = s3 or  s1                        // s6/r22 = D
06 : 00000010110101001011100000100011;          -- sub $s7, $s6, $s4
     /    s7 = s6 - s4                          // s7/r23 = C
07 : 00000010011100100000000000011001;          -- multu $s3, $s2
     /     Lo = s3 * s2                          // LO = 2D
08 : 00001000000000000000000000001000;          -- j  8       / infinite loop
09 : 00000000000000000000000000000100;                -- 4
0A : 00000000000000000000000000000101;                -- 5
```

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed

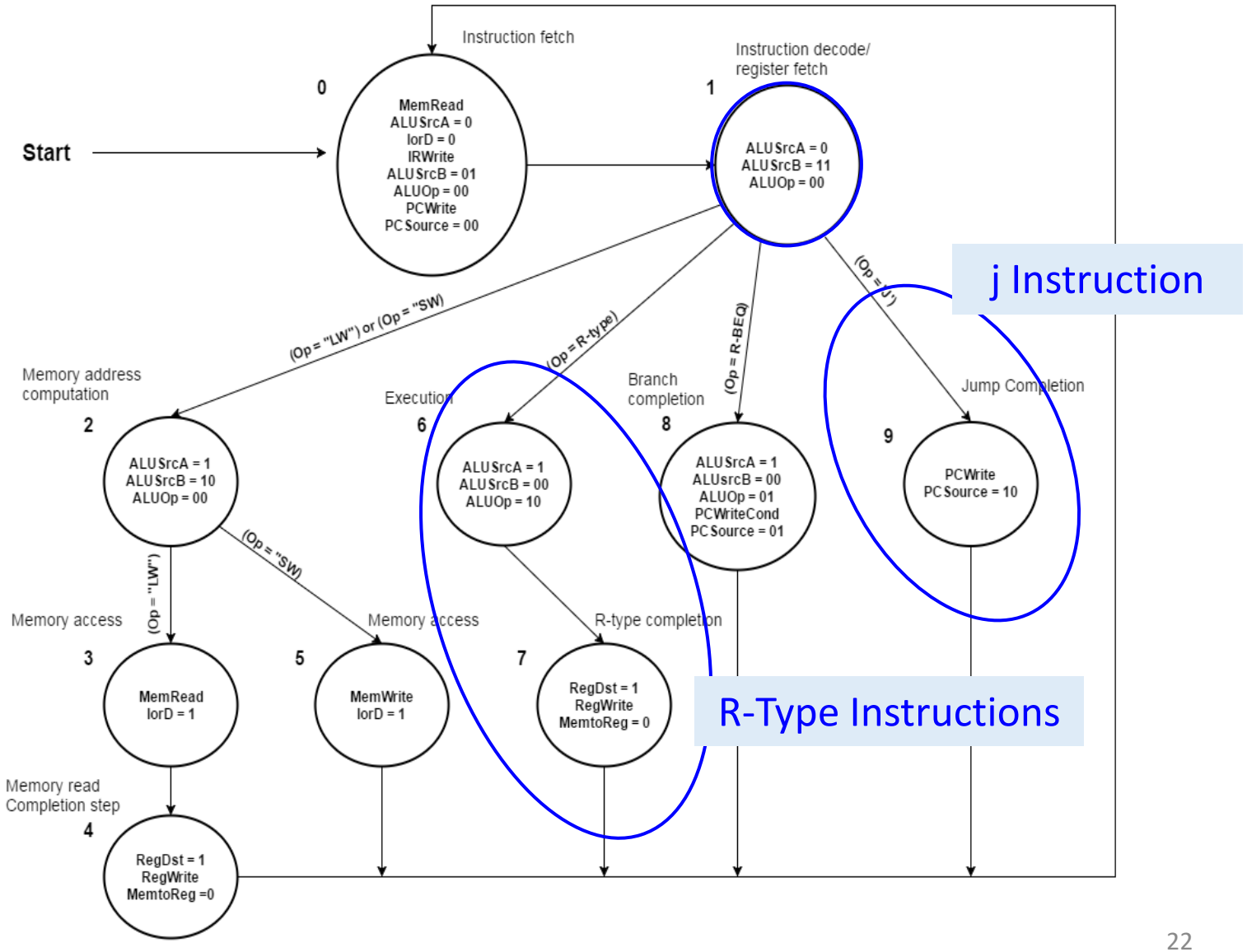
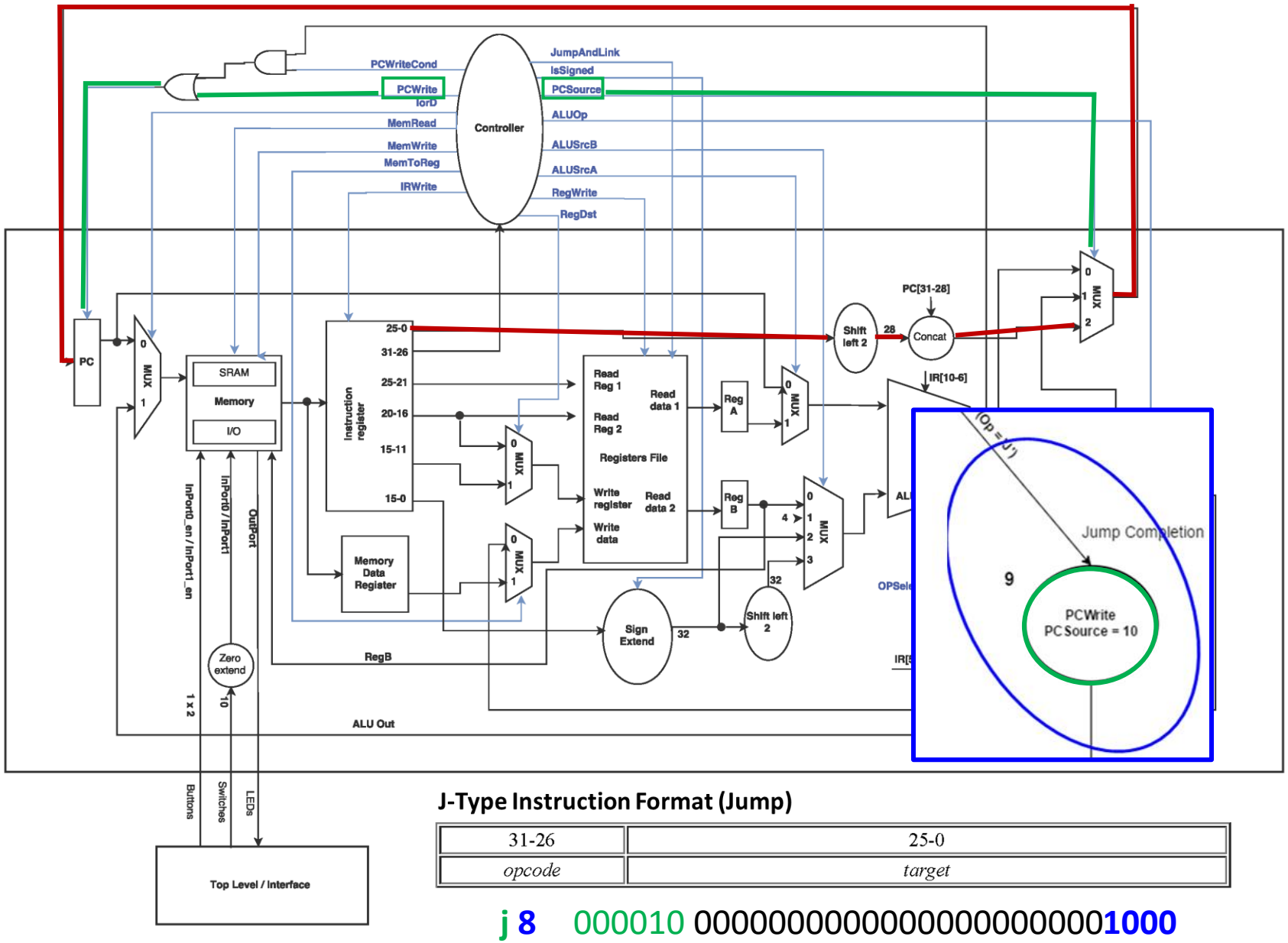


Illustration of j instruction



J-Type Instruction Format (Jump)

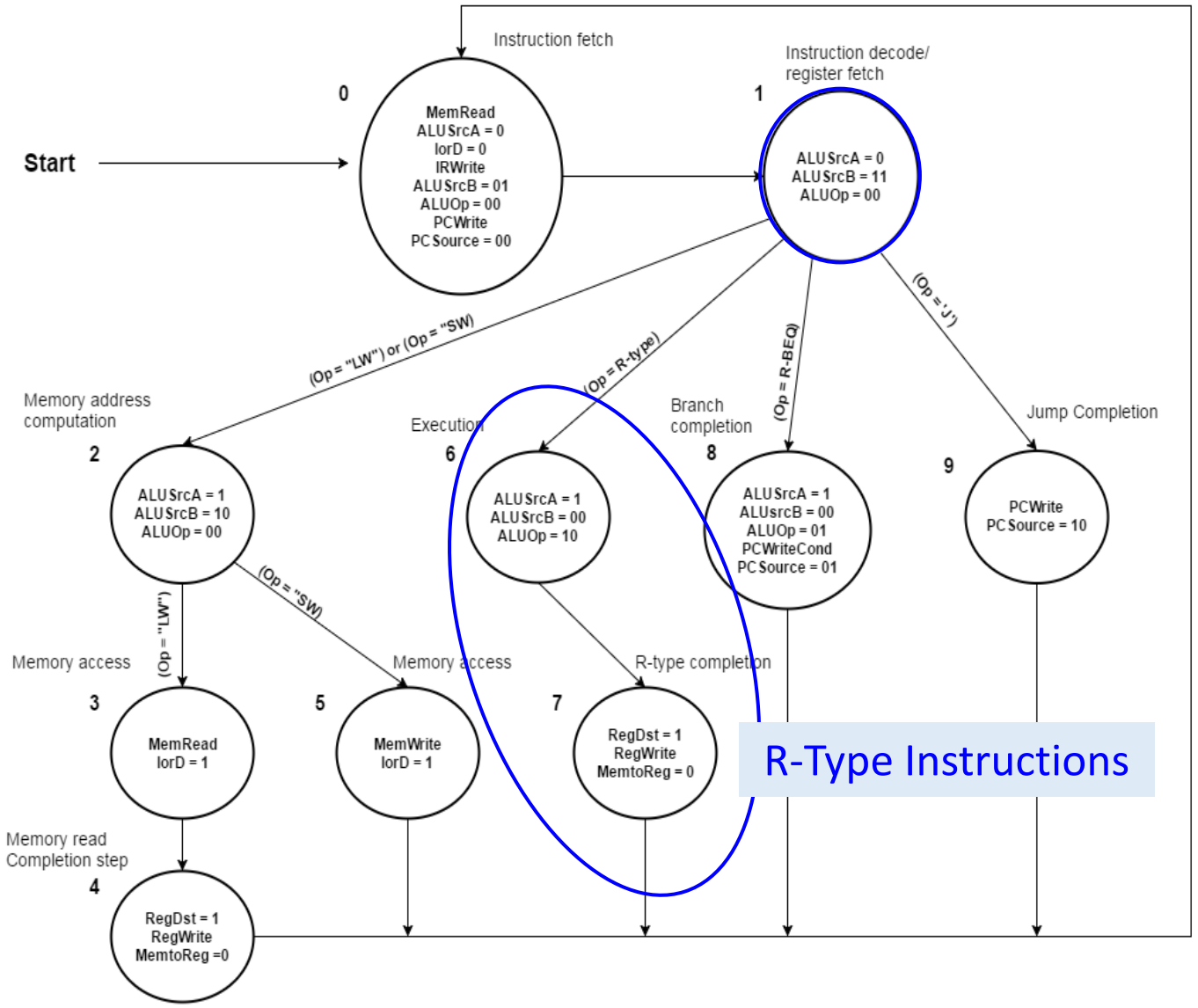
31-26	25-0
opcode	*target*

j 8 000010 00000000000000000000**1000**

- Effective addr = PC[31-28]^target^00

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed



R-Type Instruction Format

31-26	25-21	20-16	15-11	10-6	5-0
opcode (000000)	*rs*	*rt*	*rd*	*shamt*	*funct*

- ***rs*** and ***rt*** are operand registers; ***rd*** is the destination register.
- **R-Type:** Register-based ALU operations.
- ***opcode*** is always 000000.
- The actual instruction code is in the ***funct*** field (5-0).
- ***shamt*** is the "shift amount" for shift instructions.
- MIPS-4712 R-type instructions: (See MIPS instruction Set Manual for details):
  add, addu, and, mfhi, mflo, mult, multu, or, sll, sltu, sra, srl, sub, subu, xor
- Example: and $s4, $s2, $s3 (s4 = s2 and s3) or (reg20 = reg18 and reg19)
  000000 10010 10011 10100 00000 100100

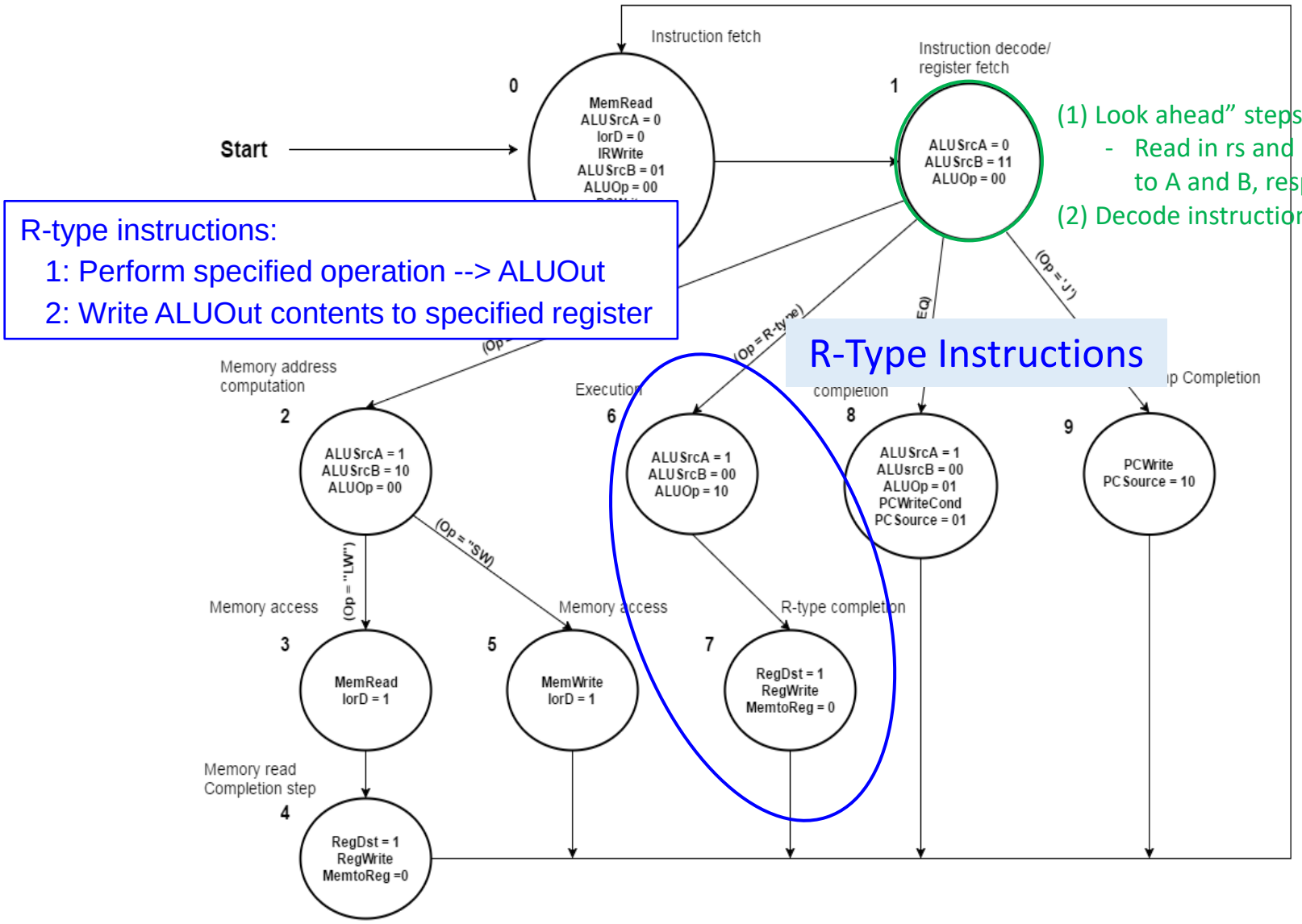
I-Type Instruction Format (ALU immediate)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*immed.*

- ***rs*** is operand A register and ***rt*** is destination register.
- ***immed.*** is operand B.
- MIPS-4712 I-type ALU insts.:addiu, andi, ori, "subiu" (does not exist), xori
- Example: addiu $s2, $s1, 7 (s2 = s1 + 7) or (reg18 = reg17 + 7)
  001100 10011 01010 000000000000111

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed



R-type instructions:

1: Perform specified operation --> ALUOut

2: Write ALUOut contents to specified register

R-Type Instructions

(1) Look ahead" steps:

- Read in rs and rt registers to A and B, respectively.

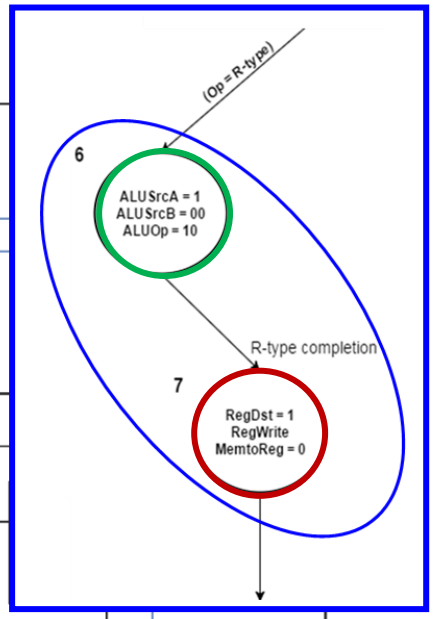
(2) Decode instruction

R-type instructions:

1: Perform specified operation --> ALUOut

2: Write ALUOut contents to specified register

n of R-type instructions

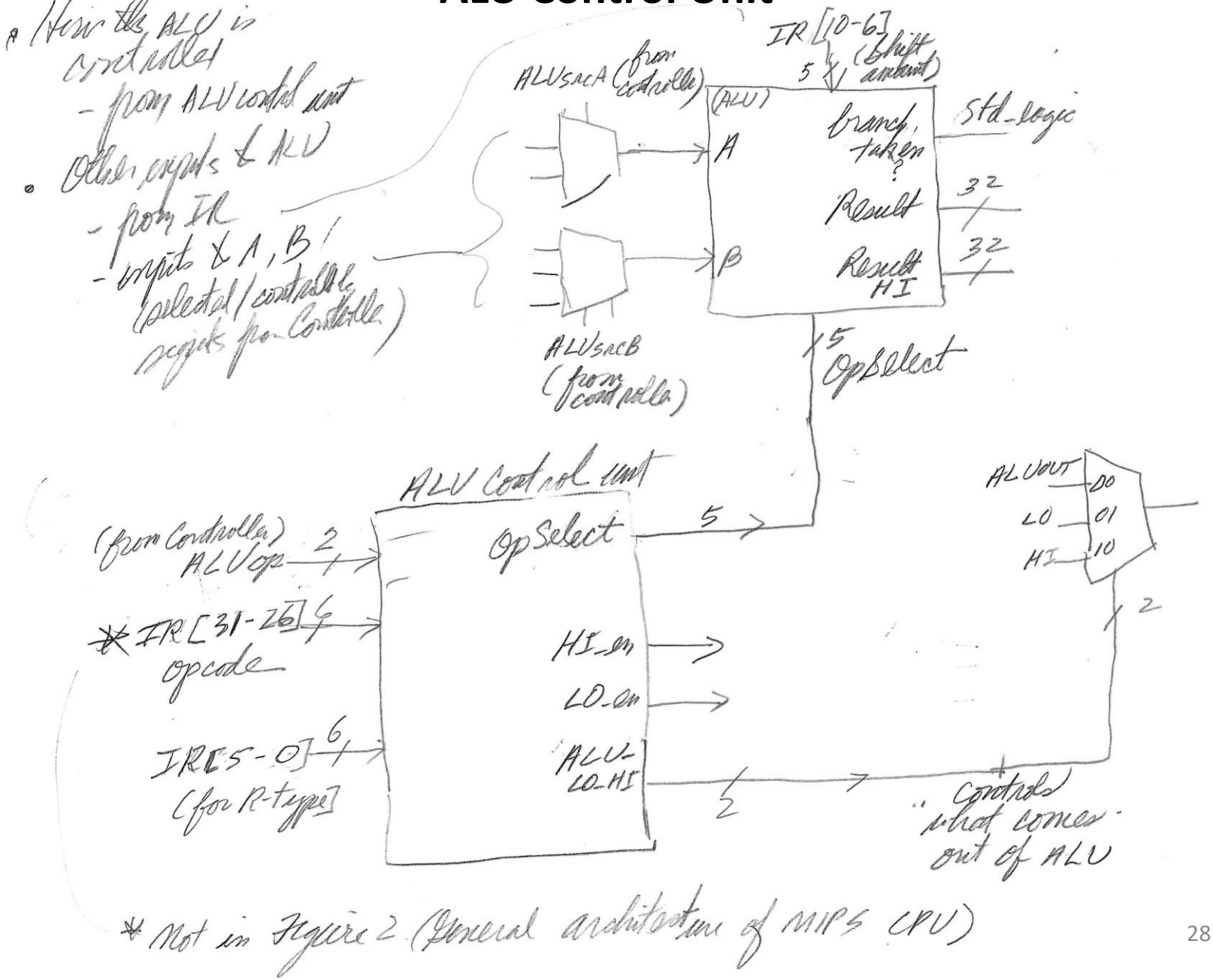


31-26	25-21	20-16	15-11	10-6	5-0
opcode (000000)	*rs*	*rt*	*rd*	*shamt*	*funct*

addu $s3, $s1, $s2 s3 = s1 + s3 (reg19 = reg17 + reg18)

000000 10001 10010 10011 00000 100001

ALU Control Unit

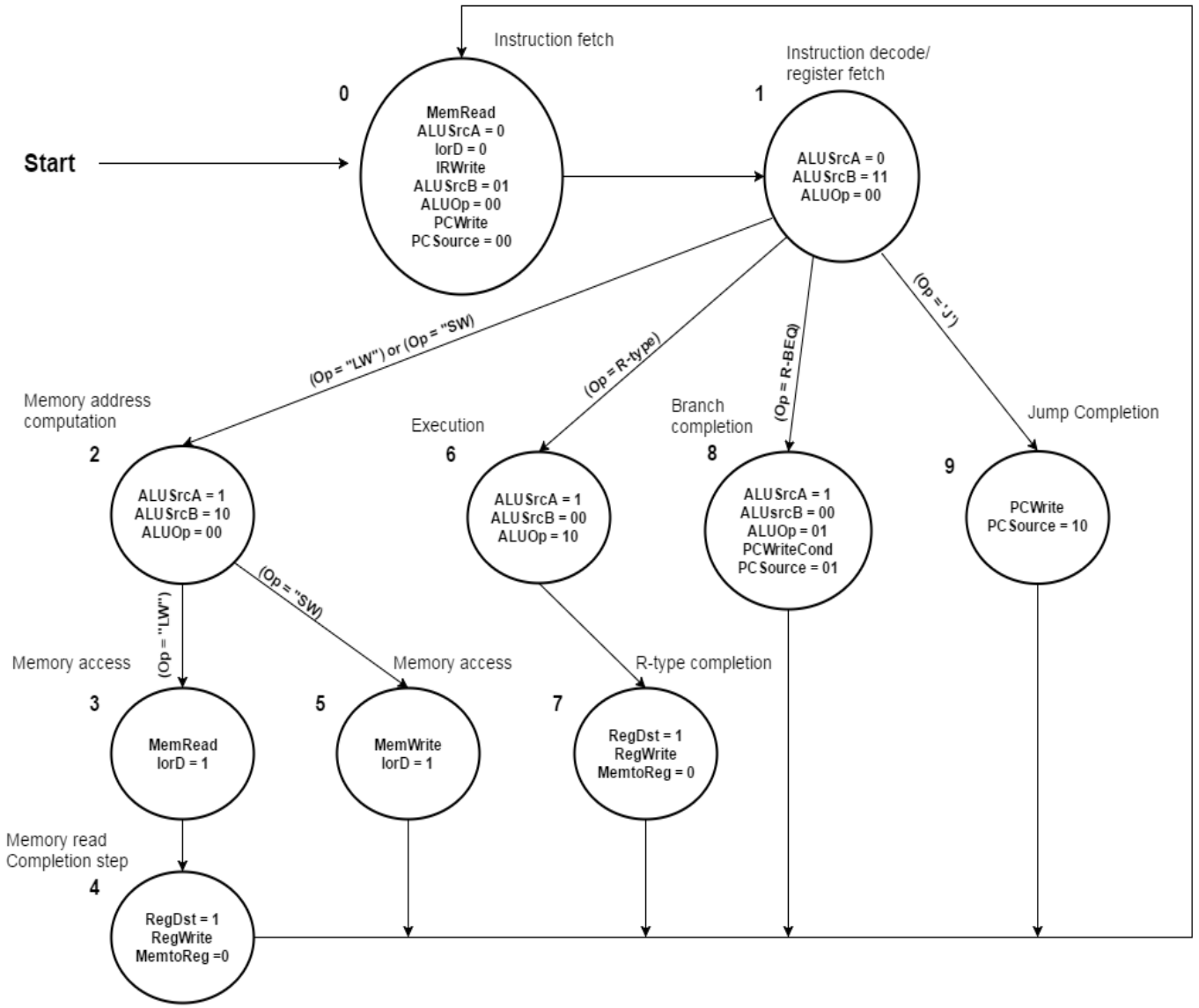


ALU Control Unit ("algorithm" for OpSelect output)

```
IF ALUOp = 00 (for State 0 (fetch), State 1 (decode), State 2)
    ALU.OpSelect → "add"
ELSIF ALUOp = 10 (State 6 R-type)
    CASE IR[5-0] (for the actual operation)
        WHEN "100001"
            ALU.OpSelect → "addu"
        WHEN "100011"
            ALU.OpSelect → "subu"
            :
        etc. for all R-type instructions
ELSE (for all the other instructions using opcode field)
    CASE IR[31-26] (Opcode in IR register)
        WHEN "001001"
            ALU.OpSelect → "addiu"
        WHEN "001100"
            ALU.OpSelect → "andi"
            :
        WHEN "etc"
            ALU.OpSelect → "etc."
                other I-type
                branch
```

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed



Upcoming Lectures

- Go over Testcase1.mif
  - Discuss each assembly instruction and the results (in memory or registers)
- Use Testcase 1 to demonstrate the tutorial on how to view RAM and Register File contents in ModelSim:
  - Start from “scratch” and create new ModelSim project, include all .vhd files, compile, simulate
  - Follow tutorial to show contents of RAM and Register File using Testcase 1.
- Demonstrate how to see signals of internal components in the Wave window. For example:
  - Cannot see “LO” register using tutorial for Testcase1.mif.
  - If simulation is incorrect in some of the instructions, may want to “trace” through the content of a register when each instruction is being executed.
- Go through rest of the testcases:
  - Testcase2.mif
  - Testcase4.mif
  - Testcase7.mif
  - Deliverable4.mif
- Assemble GCD assembly code to machine code
- Branch (BEQ, BNE) implementation
- Discuss the implementation (not illustrate in detail like before) of:
  - J and JAL instructions (TestCase7)
  - Other instructions in TestCase7 (lw and sw to ports)

TestCase7.mif (MIPS assembly code **to machine code**)

```
00 : 00001100000000000000000000000010;  -- jal  2   / jump to address 2; $ra/r31 = 4
01 : 00001000000000000000000000000001;  -- j  1      / infinite loop
02 : 00000000000000001000100000100001;  -- addu $s1, $zero, $zero
03 : 10001110001100100000000000101000;  -- lw $s2, 0A($s1)
04 : 00010010010000000000000000000011;  -- beq $s2, $zero, 8
05 : 00100110001100010000000000000100;  -- addiu $s1, $s1, 4
06 : 00000010011100101001100000100001;  -- addu   $s3, $s3, $s2
07 : 00001000000000000000000000000011;  -- j 3
08 : 10101100000100111111111111111100;  -- sw $s3, FFFC($zero)
09 : 00000011111000000000000000001000;  -- jr $ra
0A : 00000000000000000000000000000001;  -- 1
0B : ... etc. data
```

R-Type Instruction Format

31-26	25-21	20-16	15-11	10-6	5-0
opcode (000000)	*rs*	*rt*	*rd*	*shamt*	*funct*

I-Type Instruction Format (Load or Store)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

I-Type Instruction Format (Branch)

31-26	25-21	20-16	15-0
opcode	*rs*	*rt*	*address*

J-Type Instruction Format (Jump)

31-26	25-0
opcode	*target*

Figure 3. General algorithm for designing a controller for a MIPS CPU

Source: Patterson and Hennessy, Computer Organization and Design: The Hardware/Software Interface, 3rd ed

Format: `BNE rs, rt, offset`

Description: `if rs ≠ rt then branch`

(1) Look ahead" steps: (Review in Slides 16 & 17)

- Read in rs and rt registers to A and B, respectively.
- Compute target branch address using lower 16 bits of instruction --> ALUOut

(2) Decode instruction

Branch Instructions

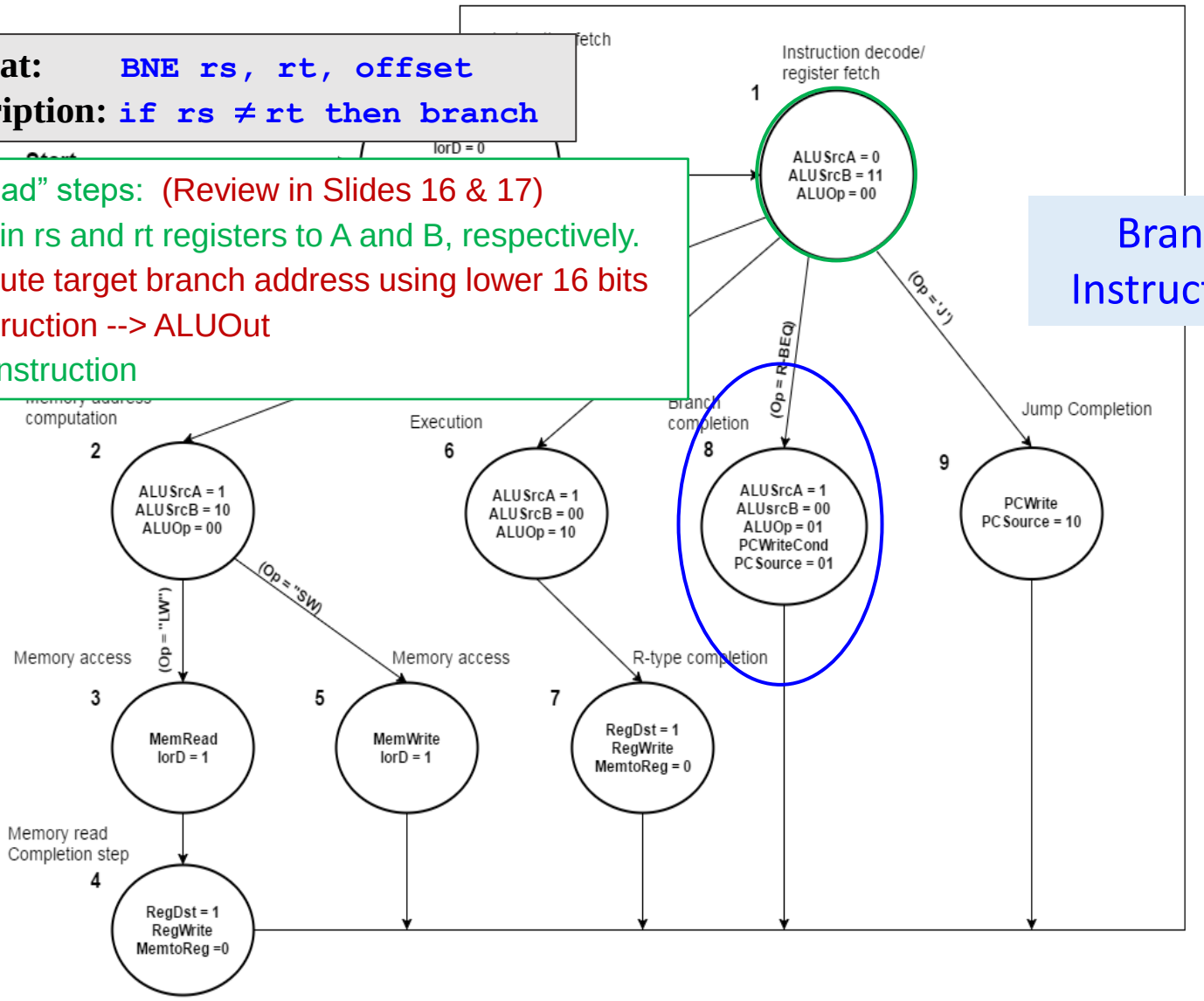
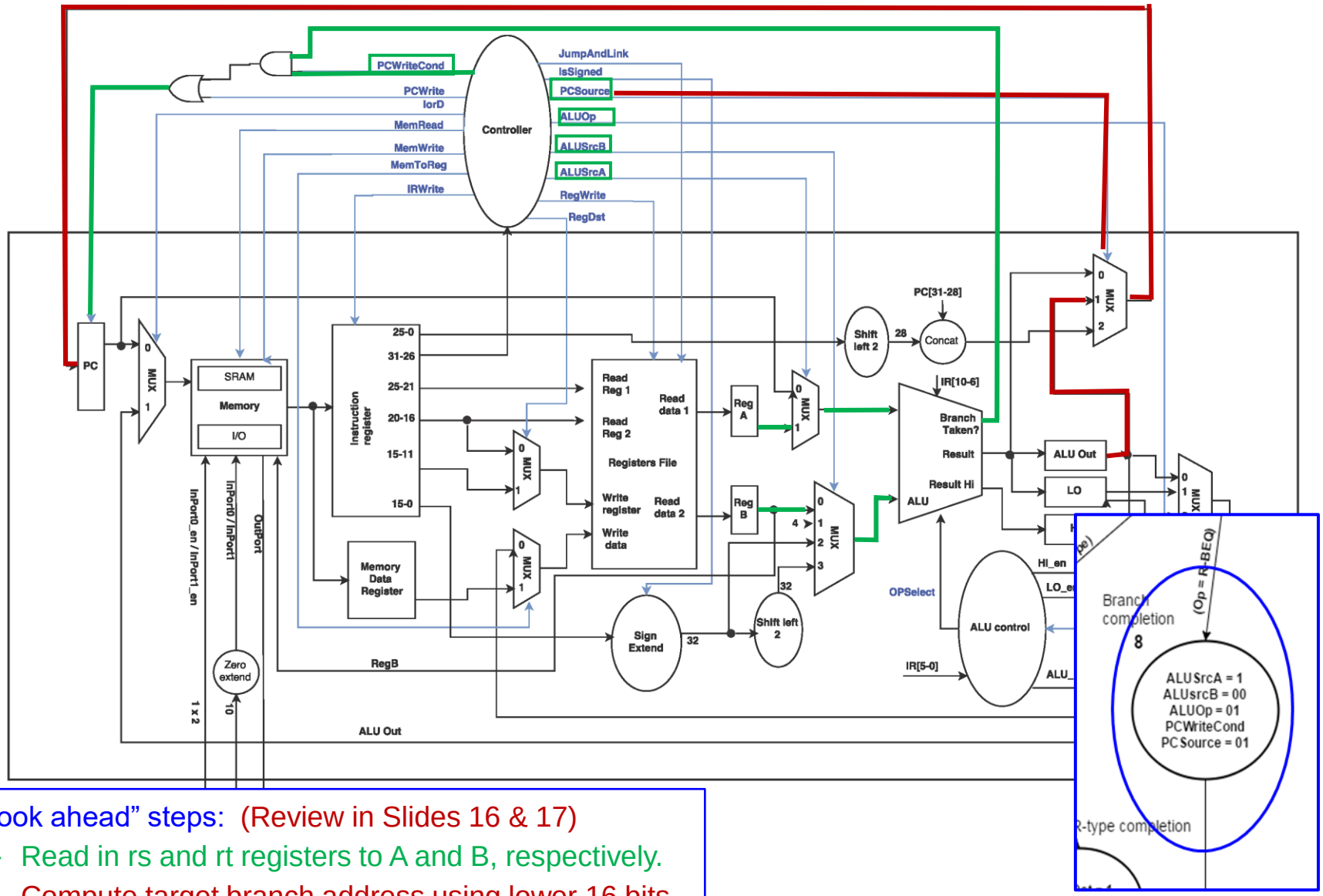


Illustration of a branch (e.g., bne) instruction



(1) Look ahead" steps: (Review in Slides 16 & 17)

- Read in rs and rt registers to A and B, respectively.
- Compute target branch address using lower 16 bits of instruction --> ALUOut

(2) Decode instruction

Format: `BNE rs, rt, offset`

Description: `if rs ≠ rt then branch`

Upcoming Lectures

- Go over Testcase1.mif
  - Discuss each assembly instruction and the results (in memory or registers)
- Use Testcase 1 to demonstrate the tutorial on how to view RAM and Register File contents in ModelSim:
  - Start from "scratch" and create new ModelSim project, include all .vhd files, compile, simulate
  - Follow tutorial to show contents of RAM and Register File using Testcase 1.
- Demonstrate how to see signals of internal components in the Wave window. For example:
  - Cannot see "LO" register using tutorial for Testcase1.mif.
  - If simulation is incorrect in some of the instructions, may want to "trace" through the content of a register when each instruction is being executed.
- Go through rest of the testcases:
  - Testcase2.mif
  - Testcase4.mif
  - Testcase7.mif
  - Deliverable4.mif
- Assemble GCD assembly code to machine code
- Branch (BEQ, BNE) implementation
- Discuss the implementation (not illustrate in detail like before) of:
  - JAL and JR instructions (TestCase7) – see MIPS manual.
  - Other instructions in TestCase7 (lw and sw to ports)