The processor

From the instruction set to the processor architecture

1

Resources and technologies

2

-

An example of instructions

The memory-reference instructions:

- *load register unscaled* (LDUR), and
- *store register unscaled* (STUR)
- LDUR X1,[X2,offset_value] or STUR X1,[X2,offset_value]

The arithmetic-logical instructions

- ADD,SUB,AND, and ORR
- ADD X1, X2, X3

The branch instructions

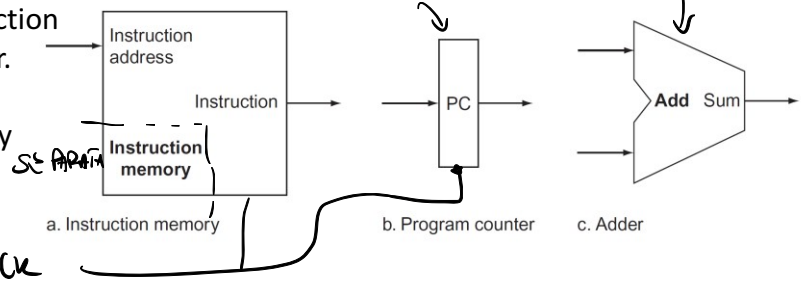
- compare and branch on zero (CBZ), and
- branch (B)

3

Read the instructions from memory

The state elements are the instruction memory and the program counter.

The instruction memory need only provide read access because the data path does not write instructions.

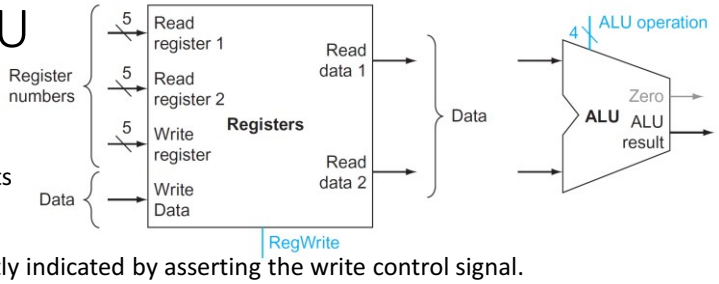


The output at any time reflects the contents of the location specified by the address input, and no read control signal is needed.

The program counter is a register that is written at the end of every clock cycle and thus does not need a write control signal.

4

The execution of ALU instructions: register file and ALU

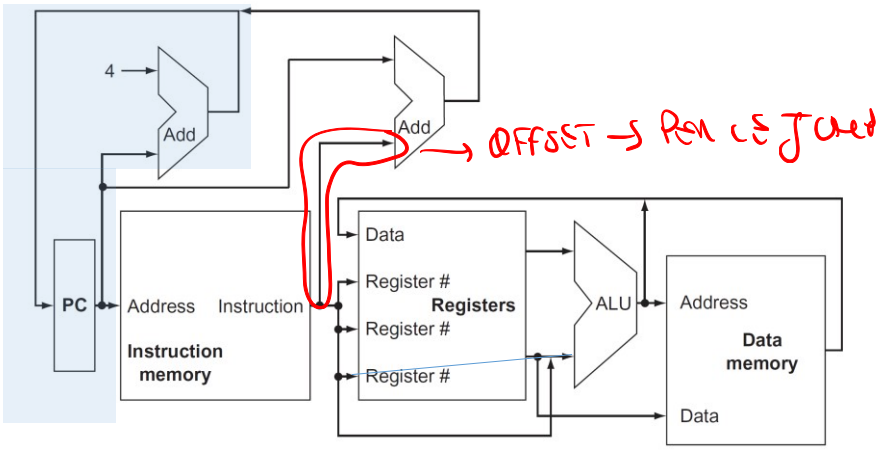


- The multiported register file contains all the registers and has two read ports and one write port.
- The register file always outputs the contents of the registers corresponding to the Read register inputs on the outputs; no other control inputs are needed.
  In contrast, a register write must be explicitly indicated by asserting the write control signal.
- The writes are edge-triggered, so that all the write inputs must be valid at the clock edge.
- Since writes to the register file are edge-triggered, our design can legally read and write the same register within a clock cycle: the read will get the value written in an earlier clock cycle, while the value written will be available to a read in a subsequent clock cycle.
- The inputs carrying the register number to the register file are all 5 bits wide, whereas the lines carrying data values are 64 bits wide.
- The operation to be performed by the ALU is
- controlled with the ALU operation signal, which will be 4 bits wide.
- We will use the Zero detection output of the ALU shortly to implement conditional branches.

5

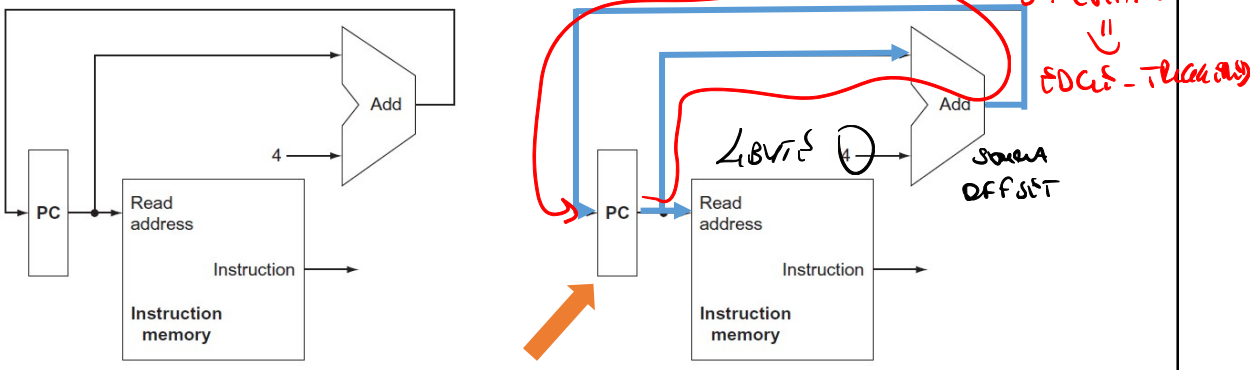
Increment PC

DOBBIAMO AUMENTARE IL PROGRAM COUNTER DOPO AVERE ESEGUITO UNA -



6

Fetching instructions and incrementing the program counter



7

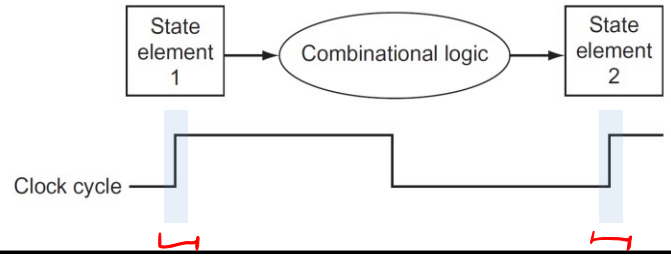
The edge-triggered clocking methodology

A **clocking methodology** defines when signals can be read and when they can be written. It is important to specify the timing of reads and writes.

The **edge-triggered clocking** methodology means that any values stored in a sequential logic element are updated only on a clock edge, which is a quick transition from low to high or vice versa.

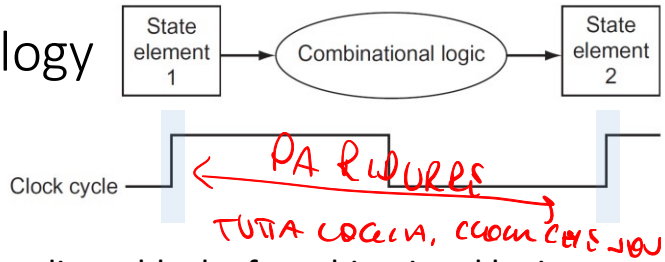
Because only state elements can store a data value, any collection of combinational logic must have its inputs come from a set of state elements and its outputs written into a set of state elements.

The inputs are values that were written in a previous clock cycle, while the outputs are values that can be used in a following clock cycle.



8

Clocking methodology



Two state elements surrounding a block of combinational logic, which operates in a single clock cycle:

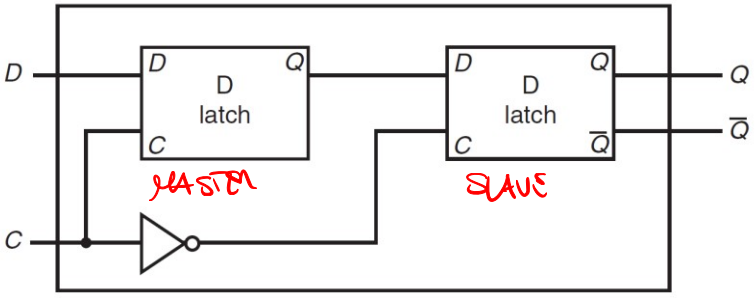
All signals must propagate from state element 1, through the combinational logic, and to state element 2 in the time ofone clock cycle.

Clock frequency

The time necessary for the signals to reach state element 2 defines the length of the clock cycle.

9

D flip-flop with a falling-edge trigger (1)



The first latch, called the master, is open and follows the input *D* when the clock input, *C*, is asserted. When the clock input, *C*, falls, the first latch is closed, but the second latch, called the slave, is open and gets its input from the output of the master latch.

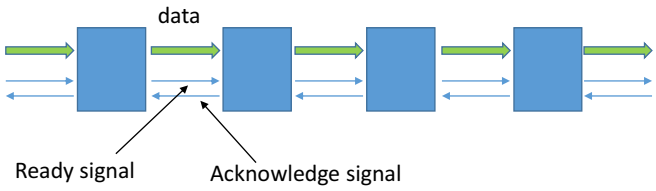
10

Pipeline data transfer

Asynchronous method

Information must be passed from one stage to the next:

- Asynchronous method



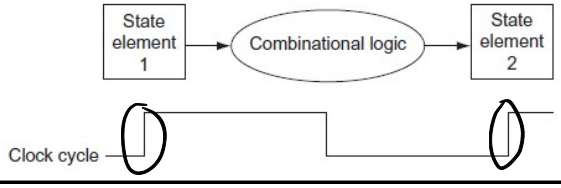
The **ready signal** informs the next unit that it is ready to pass the data.
The **acknowledge signal** is activated when the receiving unit has accepted the data.

11

11

Synchronous method

- In a synchronous digital system, the clock determines when elements with state will write values into internal storage.
- Any inputs to a state element must reach a stable value before the active clock edge causes the state to be updated.



12

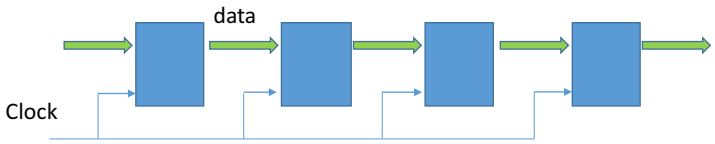
UNA PORTATA DEL CLOCK È USATA PER TRASFERIRE I DATI

12

Pipeline data transfer

Synchronous method

- Synchronous method



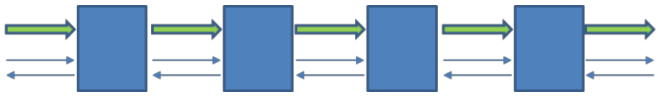
The timing signal (clock) causes all output of units to be transferred to the next units.
The timing signal occurs at fixed intervals, taking into account the slowest unit.

13

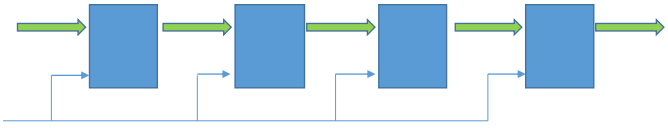
13

Pipeline data transfer

Asynchronous vs Synchronous



The asynchronous method provides the greatest flexibility in stage operating times and should make the pipeline operate at its fastest, limited dynamically by the slowest unit. The cost: the additional logic to manage the ready and acknowledge signal.

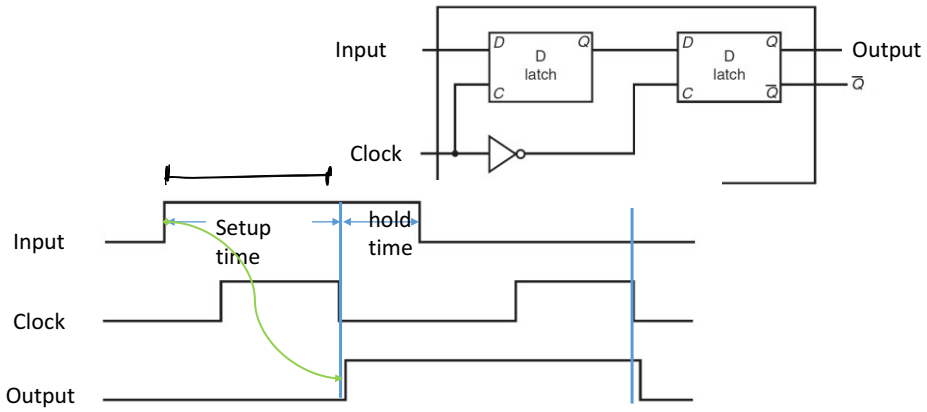


Most instruction and arithmetic pipelines use the synchronous method to reduce logic complexity and implementation problems.

14

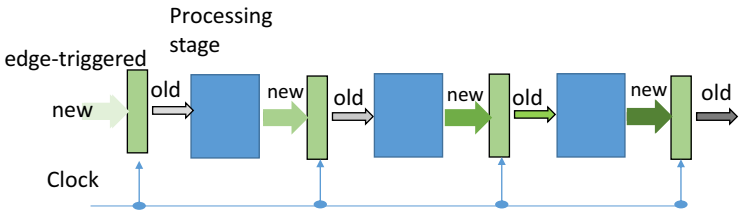
14

D flip-flop with a falling-edge trigger (2)



15

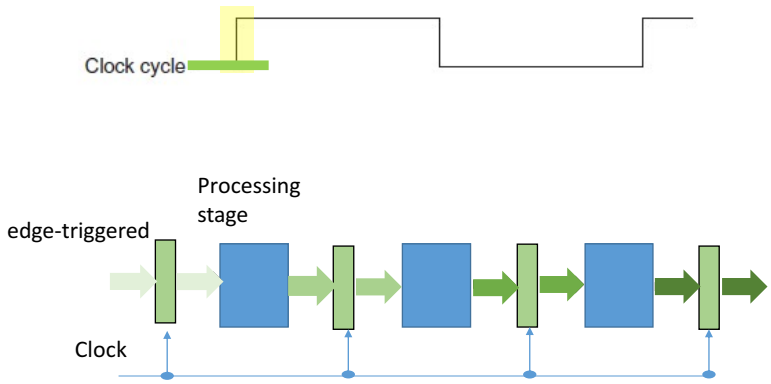
Pipeline with **edge-triggered methodology (1)**



16

16

Pipeline with **edge-triggered methodology (2)**

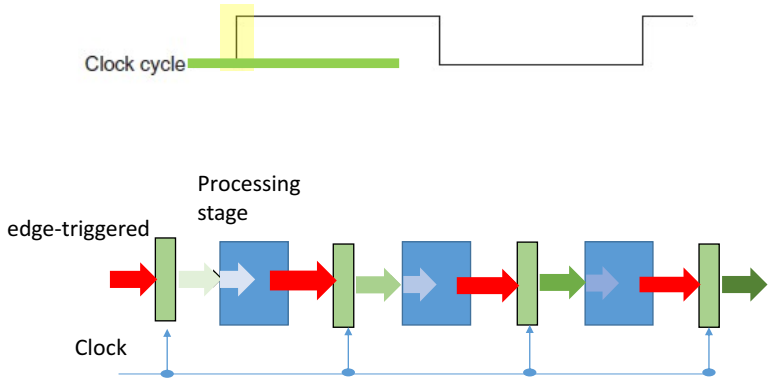


.

17

17

Pipeline with **edge-triggered methodology (3)**

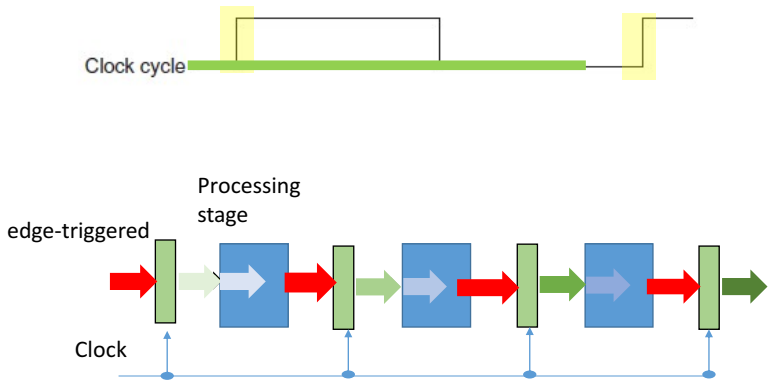


.

18

18

Pipeline with **edge-triggered methodology (4)**

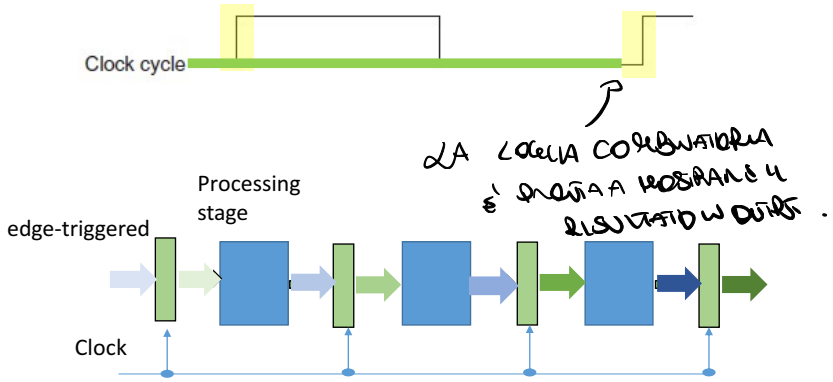


.

19

19

Pipeline with **edge-triggered methodology (5)**

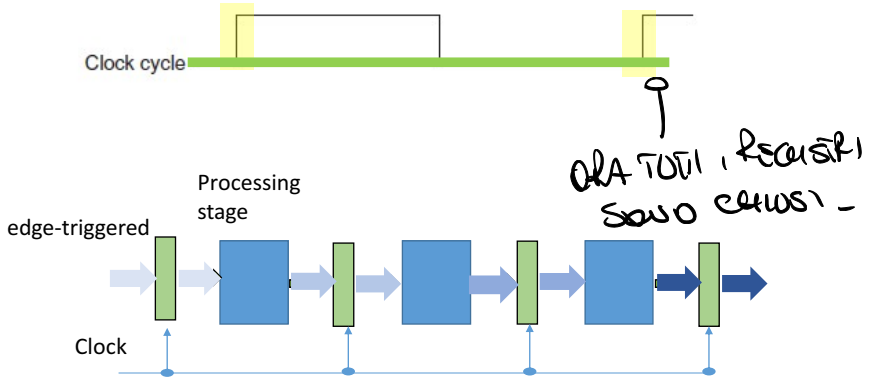


.

20

20

Pipeline with **edge-triggered methodology** (6)

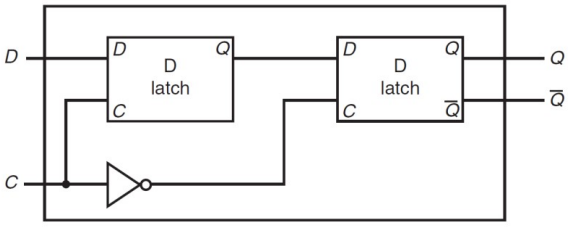


.

21

21

D flip-flop with a falling-edge trigger (1)



The first latch, called the master, is open and follows the input *D* when the clock input, *C*, is asserted. When the clock input, *C*, falls, the first latch is closed, but the second latch, called the slave, is open and gets its input from the output of the master latch.

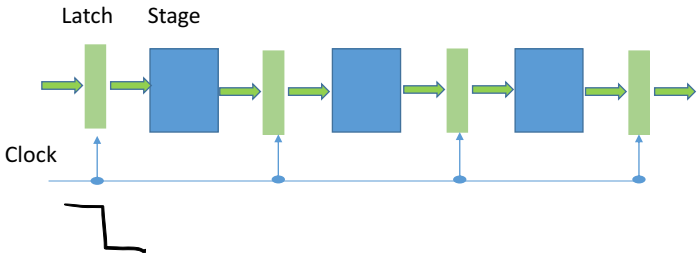
22

Pipeline data transfer

Synchronous method

An edge-triggered methodology allows us to read the contents of a register, send the value through some combinational logic, and write that register in the same clock cycle.

- Pipeline with staging latches

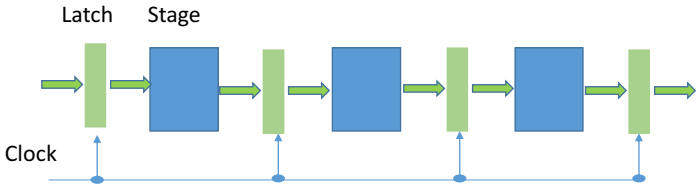


The clock signal activates all the staging latches simultaneously.

23

23

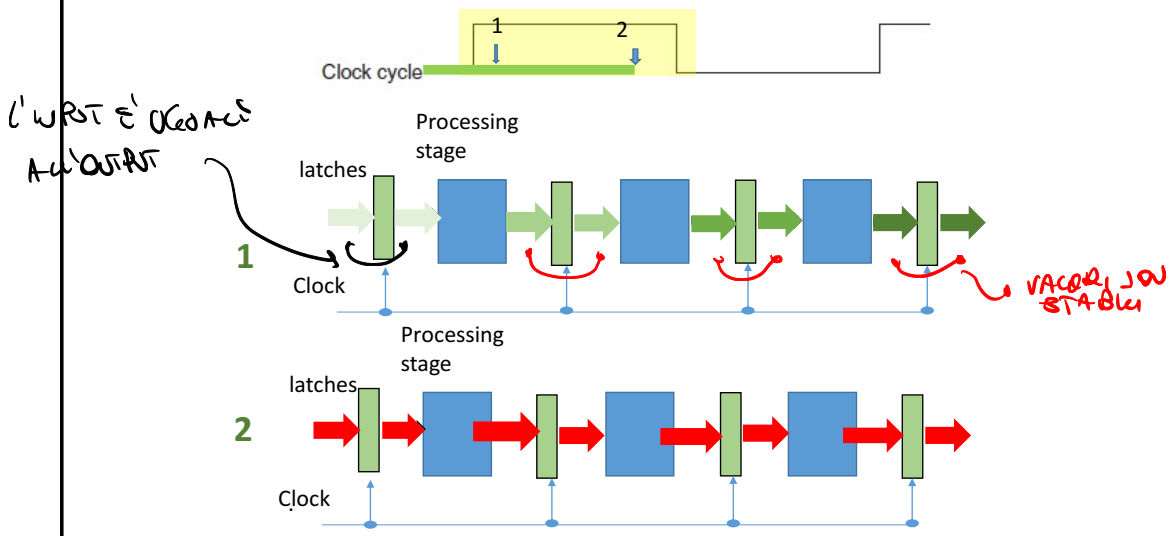
Pipeline with staging latches



24

24

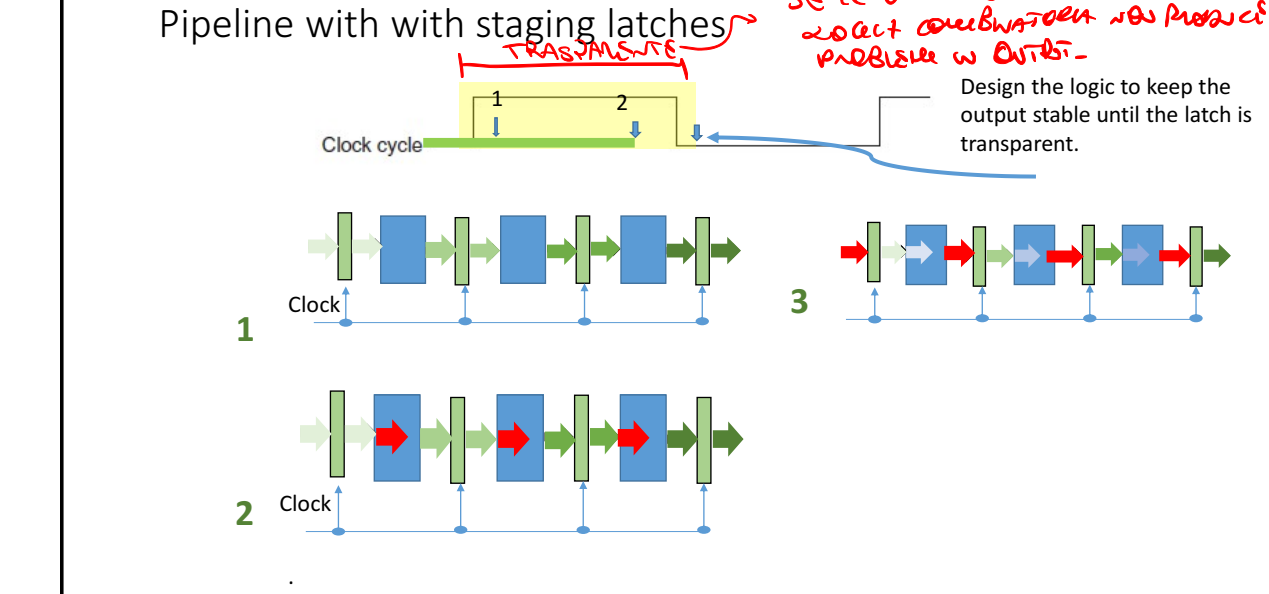
Pipeline with with staging latches: the problem



25

25

Pipeline with with staging latches

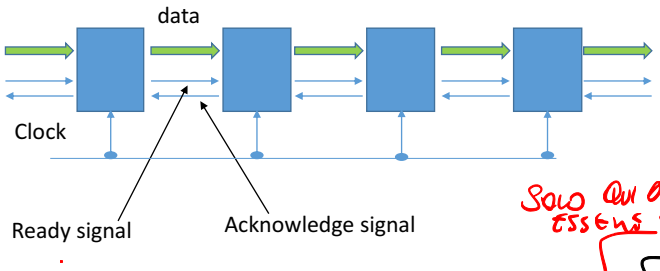


26

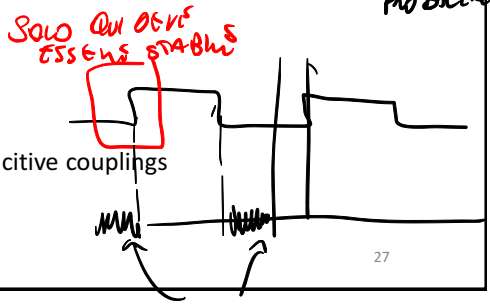
26

Data transfer

Asynchronous method with clock (synchronized)



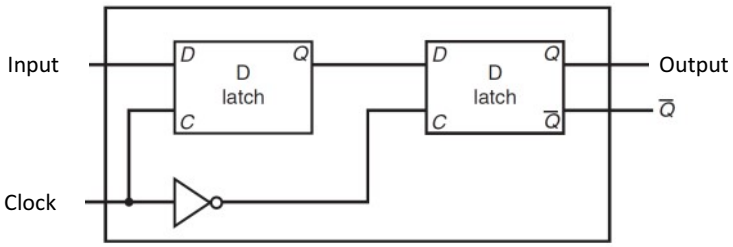
Used in industrial plants to reduce the effects of capacitive couplings and electromagnetic disturbances on the transfer.

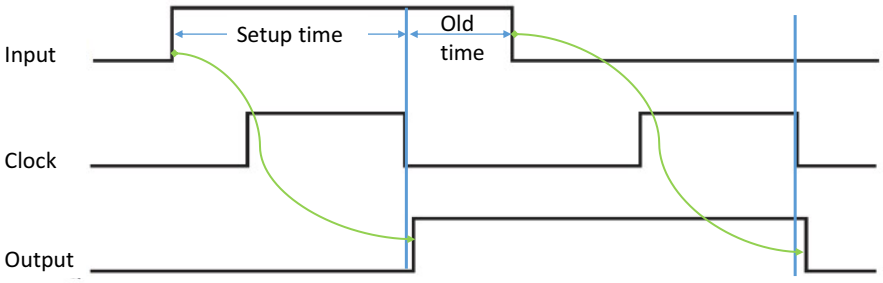


27

27

D flip-flop with a falling-edge trigger (2)





28

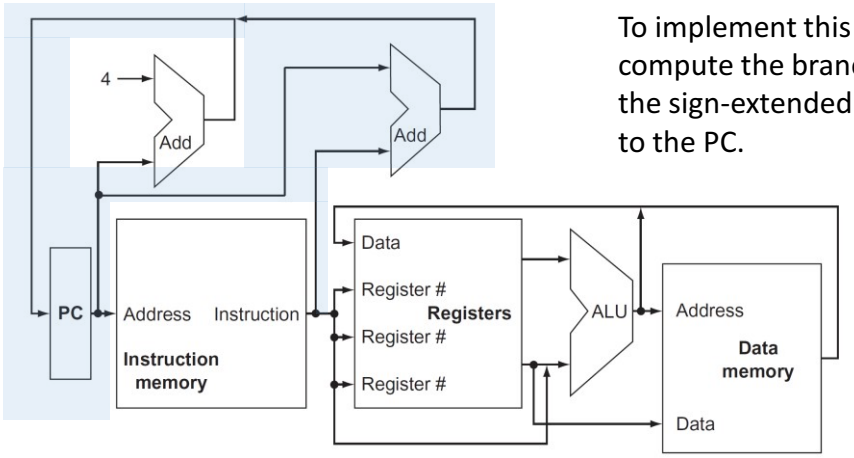
Resources and instructions

29

Branch
CBZ X1,offset

The CBZ instruction has two operands, a register that is tested for zero, and a 19-bit offset used to compute the **branch target address** relative to the branch instruction address

To implement this instruction, we must compute the branch target address by adding the sign-extended offset field of the instruction to the PC.



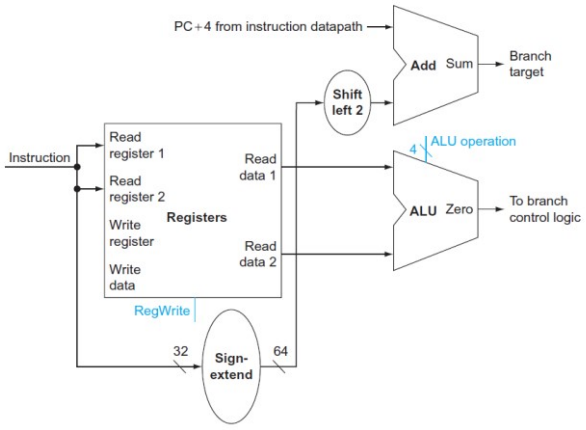
The architecture also states that the offset field is shifted left 2 bits so that it is a word offset.

This shift increases the effective range of the offset field by a factor of 4.

30

CBZ X1,offset

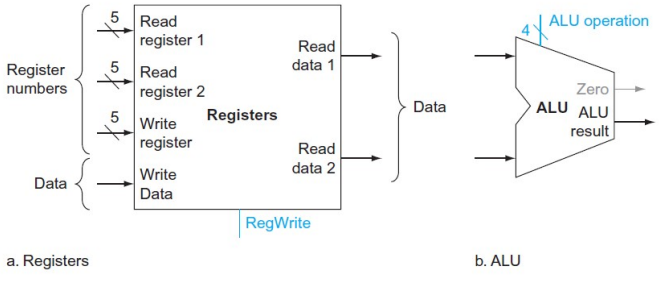
- We must also determine whether the next instruction is the instruction that follows sequentially or the instruction at the branch target address.
- When the condition is true (i.e., the operand is zero), the branch target address becomes the new PC.
- If the operand is not zero, the incremented PC should replace the current PC.
- Thus, the branch datapath must do two operations: compute the branch target address and test the register contents.



31

ADD,SUB,AND, and ORR

For each data word to be read from the registers, we need an input to the register file that specifies the *register number* to be read and an output from the register file that will carry the value that has been read from the registers.



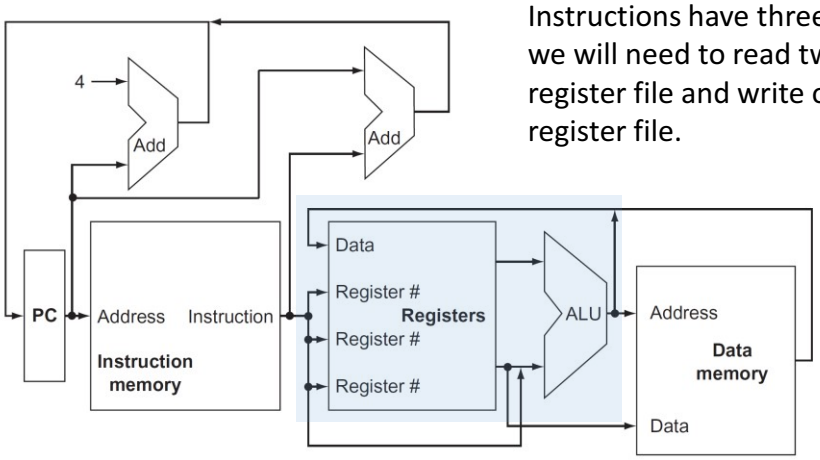
To write a data word, we will need two inputs: one to specify the register number to be written and one to supply the *data* to be written into the register.

The register file always outputs the contents of whatever register numbers are on the Read register inputs.

Writes, however, are controlled by the write control signal, which must be asserted for a write to occur at the clock edge.

32

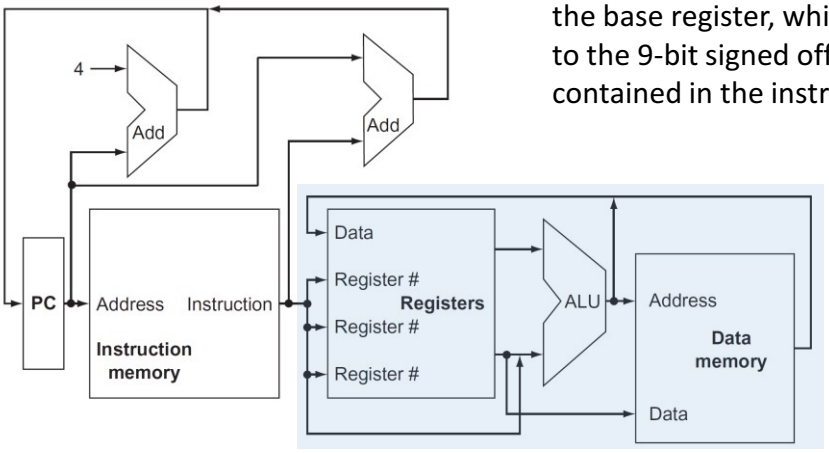
ADD X1, X2, X3



Instructions have three register operands, so we will need to read two data words from the register file and write one data word into the register file.

33

load register unscaled (LDUR)
LDUR X1,[X2,offset_value]

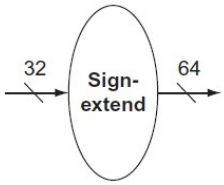


The instruction computes a memory address by adding the base register, which is X2, to the 9-bit signed offset field contained in the instruction.

34

Sign extension unit

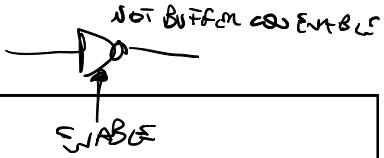
- We will need a unit to **sign-extend** the 9-bit offset field in the instruction to a 64-bit signed value.



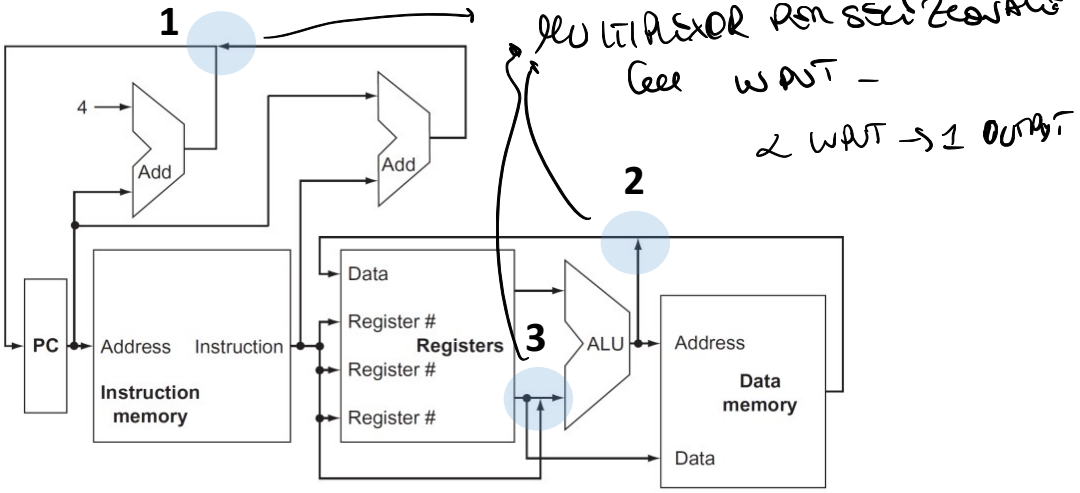
35

C'È ANCHE UN'ALTRA SOLUZIONE PER EVITARE IL MULTIPLEXER.

↳ 3-STATE GATE



A *multiplexor to select data* from two different sources

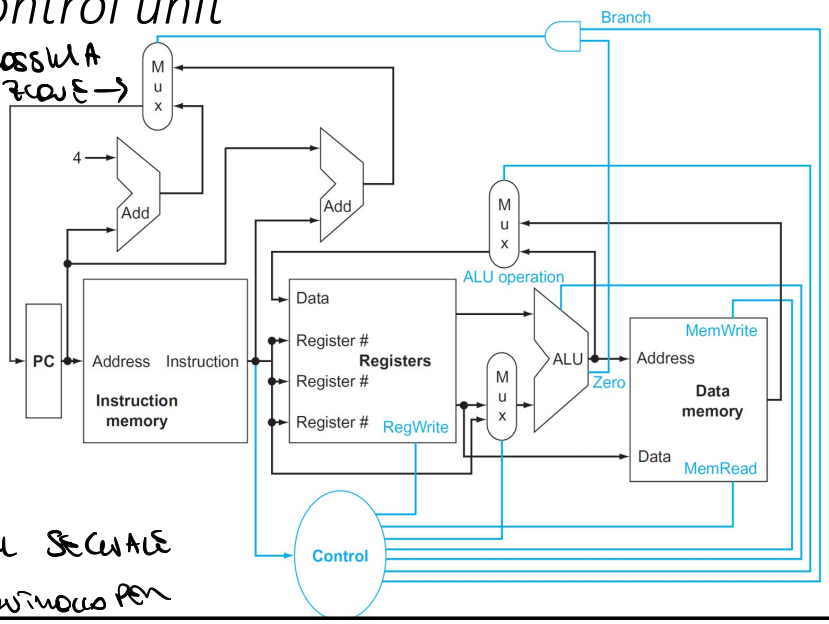


MULTIPLEXOR PER SELEZIONARE CHE WAIT - 2 WAIT → 1 OUTPUT

36

The main *control unit*

Which has the instruction as an input, is used to determine how to set the control lines for the functional units and the multiplexors.

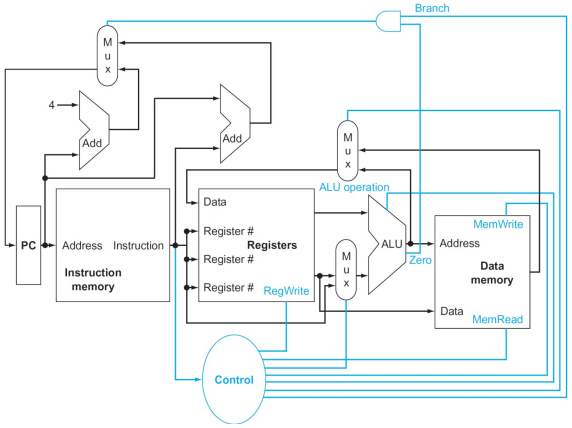


37

The ALU Control

- Depending on the instruction class, the ALU will need to perform one of these first five functions.
- For load register and store register instructions, we use the ALU to compute the memory address by addition.
- For the instructions, the ALU needs to perform one of the four actions (AND, OR, subtract, or add), depending on the value of the 11-bit opcode field in the instruction.
- For compare and branch zero, the ALU just passes the value of register input.

ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	pass input b
1100	NOR



38

The main control Unit

We can generate the 4-bit ALU control input using a small control unit that has as inputs the opcode field of the instruction and a 2-bit control field, which we call ALUOp.

ALUOp indicates whether the operation to be performed should be add (00) for loads and stores, pass input b (01) for CBZ, or be determined by the operation encoded in the opcode field (10).

Instruction fields

Output

Instruction	ALUOp	Instruction operation	Opcode field	Desired ALU action	ALU control input
LDUR	00	load register	XXXXXXXXXXX	add	0010
STUR	00	store register	XXXXXXXXXXX	add	0010
CBZ	01	compare and branch on zero	XXXXXXXXXXX	pass input b	0111
R-type	10	ADD	10001011000	add	0010
R-type	10	SUB	11001011000	subtract	0110
R-type	10	AND	10001010000	AND	0000
R-type	10	ORR	10101010000	OR	0001

39

The truth table for the 4 ALU control bits

ALUOp		Opcode field											Operation
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[24]	I[23]	I[22]	I[21]	
0	0	X	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	X	0111
1	X	1	0	0	0	1	0	1	1	0	0	0	0010
1	X	1	1	0	0	1	0	1	1	0	0	0	0110
1	X	1	0	0	0	1	0	1	0	0	0	0	0000
1	X	1	0	1	0	1	0	1	0	0	0	0	0001

40

Instruction format and paths

41

Instruction formats (I)

Field	opcode	Rm	shamt	Rn	Rd
Bit positions	31:21	20:16	15:10	9:5	4:0

The format for ADD,SUB, AND, and ORR. They have three register operands: Rn, Rm, and Rd.

Fields Rn and Rm are sources, and Rd is the destination.

ALUOp

Field	1986 or 1984	address	0	Rn	Rt
Bit positions	31:21	20:12	11:10	9:5	4:0

b. Load or store instruction

The register Rn is the base register that is added to the 9-bit address field to form the memory address.

- For loads, Rt is the destination register for the loaded value.
- For stores, Rt is the source register whose value should be stored into memory.

42

Instruction formats (II)

Field	180	address	Rt
Bit positions	31:26	23:5	4:0

c. Conditional branch instruction

Instruction format for compare and branch on zero (opcode = 180).

The register Rtis the source register that is tested for zero. The 19-bit address field is sign-extended, shifted, and added to the

- PC to compute the branch target address.

First design principle: *simplicity favors regularity* *Opcode*

The **opcode** field is between 6 and 11 bits wide and found in bits 31:26 to 31:21.

Field	opcode	Rm	shamt	Rn	Rd
Bit positions	31:21	20:16	15:10	9:5	4:0

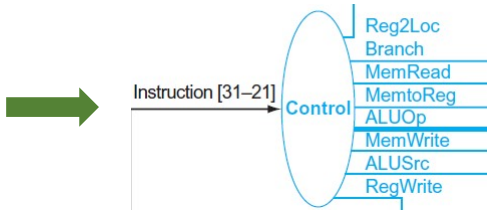
a. R-type instruction

Field	1986 or 1984	address	0	Rn	Rt
Bit positions	31:21	20:12	11:10	9:5	4:0

b. Load or store instruction

Field	180	address	Rt
Bit positions	31:26	23:5	4:0

c. Conditional branch instruction



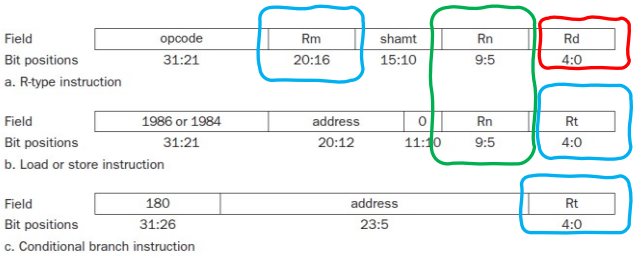
First design principle: *simplicity favors regularity* *Register operands*

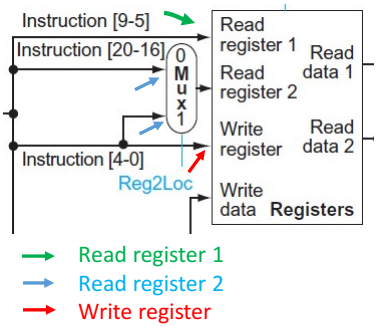
The first register operand is always in bit positions 9:5 (Rn) for both R-type instructions and for the base register for load and store instructions.

The other register operand is in one of two places. It is in bit positions 20:16 (Rm) for R-type instructions and it is in bit positions 4:0 (Rt) for the register to be written by a load.

That is also the field that specifies the register to be tested for zero for compare and branch on zero.

We will need to add a multiplexor to select which field of the instruction is used to indicate the register number to be read.



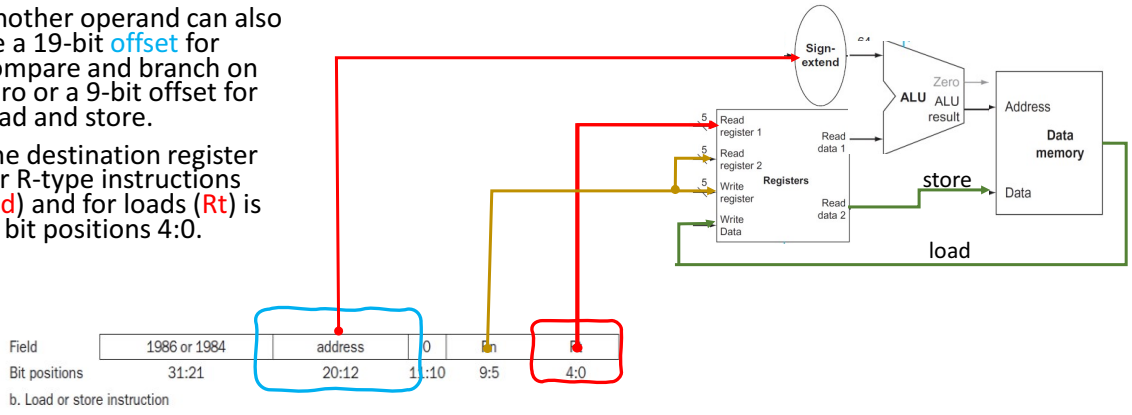


45

First design principle: *simplicity favors regularity* *Operands (I)*

Another operand can also be a 19-bit offset for compare and branch on zero or a 9-bit offset for load and store.

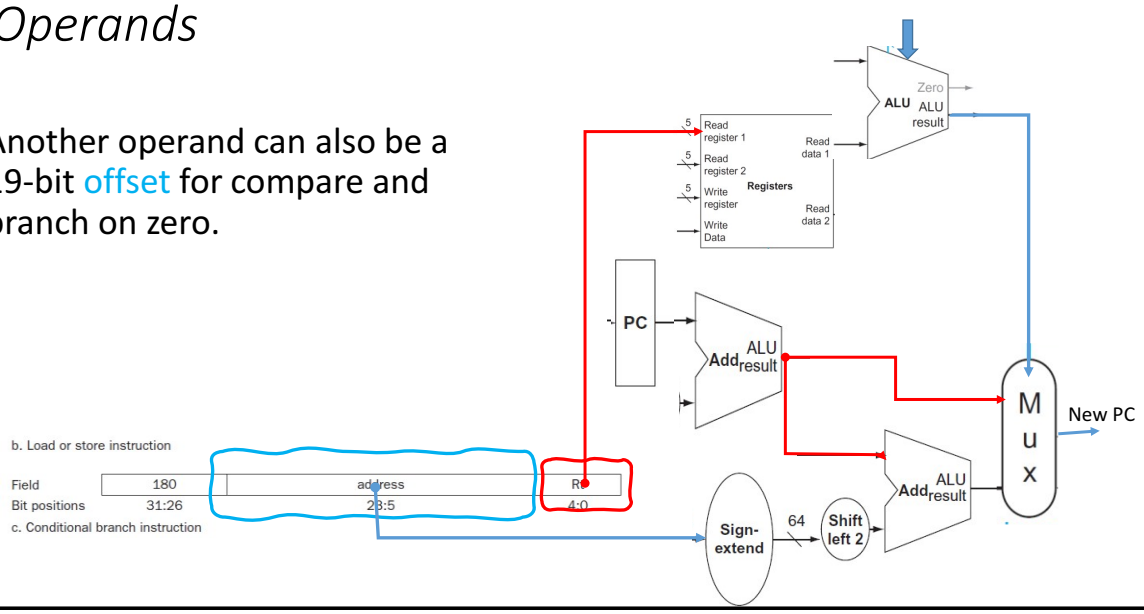
The destination register for R-type instructions (Rd) and for loads (Rt) is in bit positions 4:0.



46

First design principle: *simplicity favors regularity Operands*

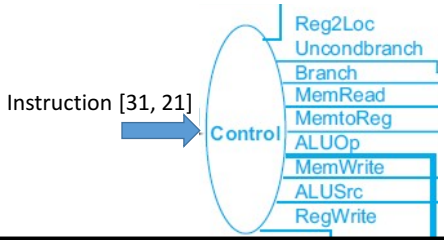
Another operand can also be a 19-bit offset for compare and branch on zero.

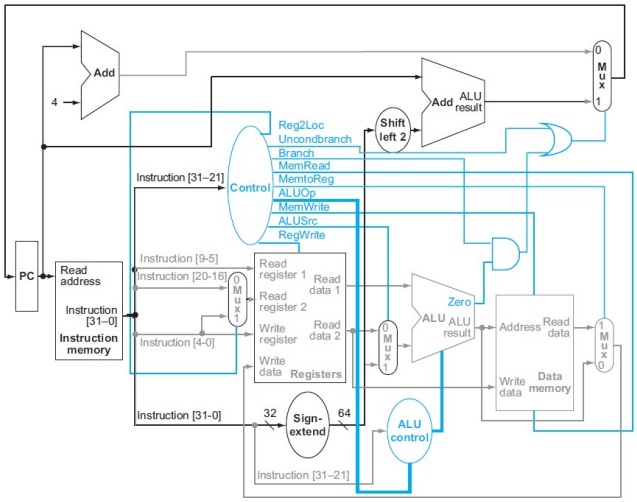


47

Signals from Main Contol Unit

Signal name	Effect when deasserted	Effect when asserted
Reg2Loc	The register number for Read register 2 comes from the Rm field (bits 20:16).	The register number for Read register 2 comes from the Rt field (bits 4:0).
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, lower 16 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.





The state elements all have the clock as an implicit input and that the clock is used in controlling writes.

48

Designing the Main Control Unit

49

Instruction execution

50

- 1. Fetch instruction from memory.
- 2. Read registers and decode the instruction.
- 3. Execute the operation or calculate an address.
- 4. Access an operand in data memory (if necessary).
- 5. Write the result into a register (if necessary).

51

ADD X1,X2,X3

Although everything occurs in one clock cycle, we can think of four steps to execute the instruction:

1. The instruction is fetched, and the PC is incremented.

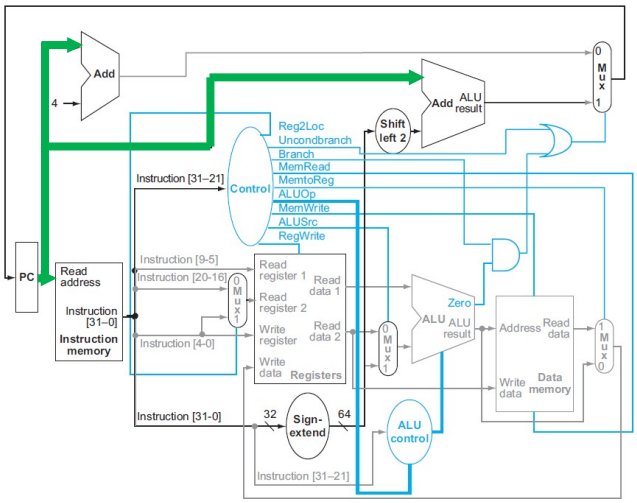
2. Two registers, X2 and X3, are read from the register file; also, the main control unit computes the setting of the control lines during this step.

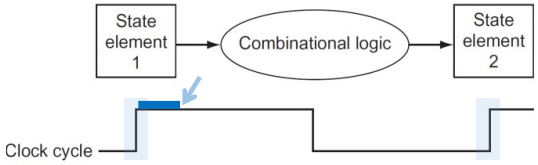
3. The ALU operates on the data read from the register file, using portions of the opcode to generate the ALU function.

4. The result from the ALU is written into the destination register (X1) in the register file.

52

ADD X1, X2, X3



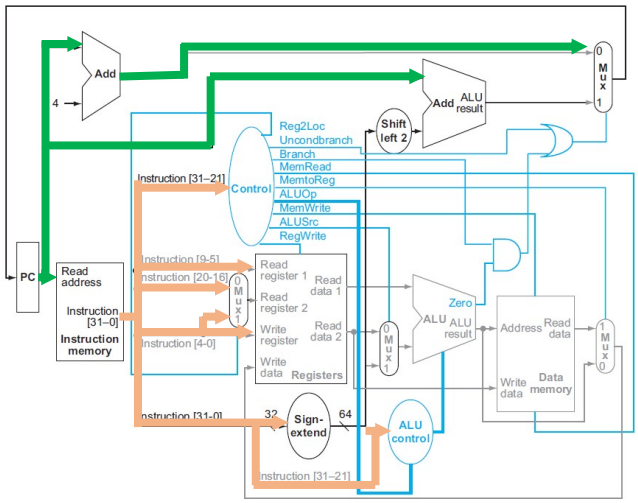


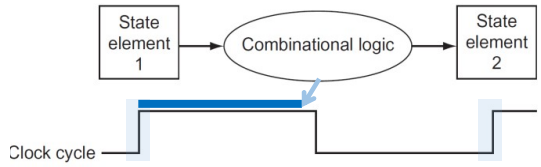
Fetch of the instruction

- 1. Send the *program counter* (PC):
  - to the memory that contains the code, and
  - to the two adders used to compute the address of the next instruction.

53

ADD X1, X2, X3





1. Fetch of the instruction

a. Send the *program counter* (PC):

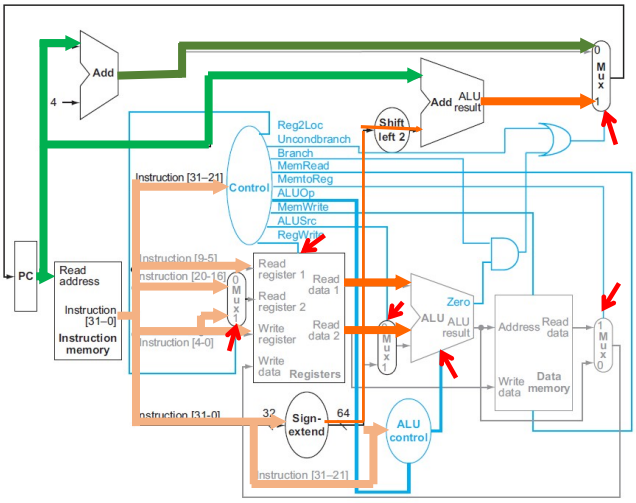
- to the memory that contains the code, and
- to the two adders used to compute the address of the next instruction.

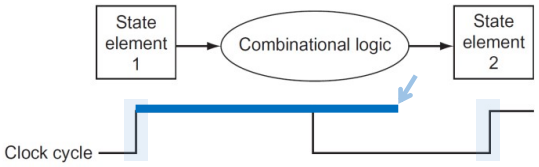
b. The received instruction is sent:

- to the register file, and
- to the control unit.

54

ADD X1, X2, X3

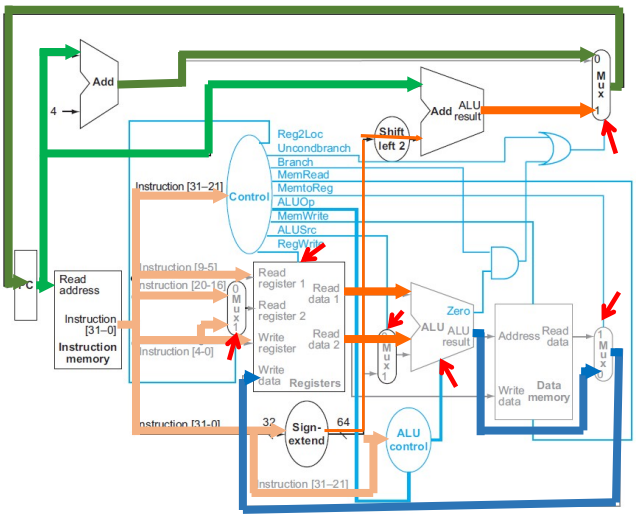


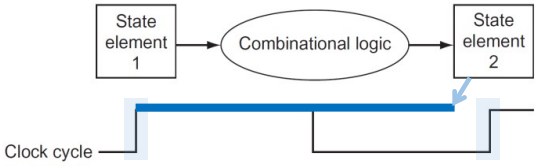


1. Fetch of the instruction
2. Read registers and decode the instruction

55

ADD X1, X2, X3

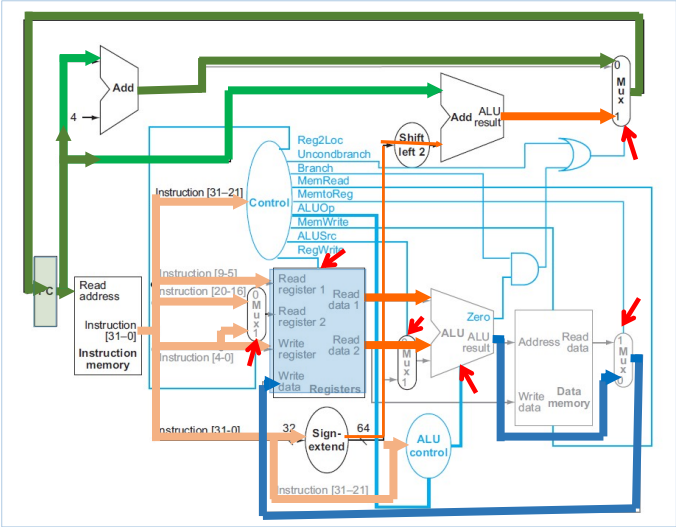


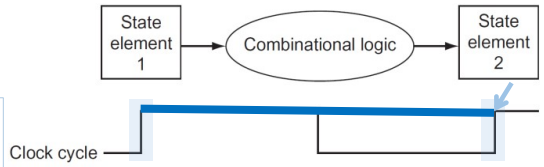


1. Fetch of the instruction and increment PC
2. Read registers and decode the instruction.
3. The ALU operates on the data read from the register file, using portions of the opcode to generate the ALU function.

56

ADD X1, X2, X3





1. Fetch of the instruction
2. Read registers and decode the instruction.
3. The ALU operates on the data read from the register file, using the opcode to generate the ALU function.
4. The result from the ALU is written into the destination register (X1).

The state elements all have the clock as an implicit input and that the clock is used in controlling writes.

57

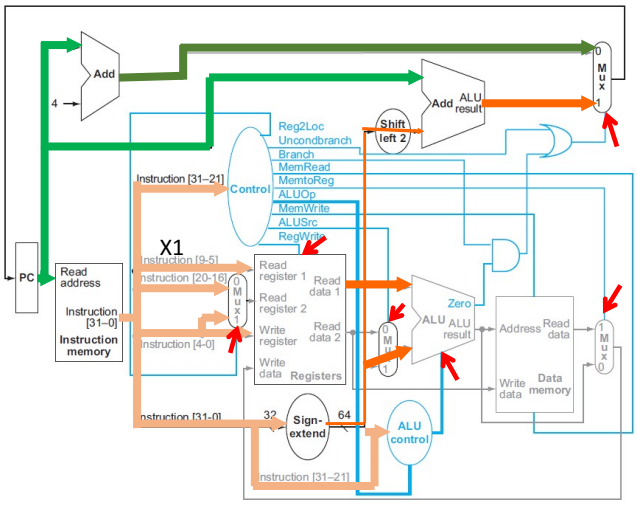
LDUR X1, [X2,offset]

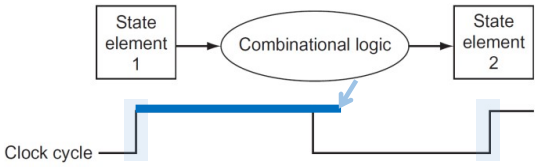
We can think of a load instruction as operating in five steps:

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. A register (X2) value is read from the register file.
3. The ALU computes the sum of the value read from the register file and the sign-extended 9 bits of the instruction (offset).
4. The sum from the ALU is used as the address for the data memory.
5. The data from the memory unit is written into the register file (X1).

58

LDUR X1, [X2,offset]

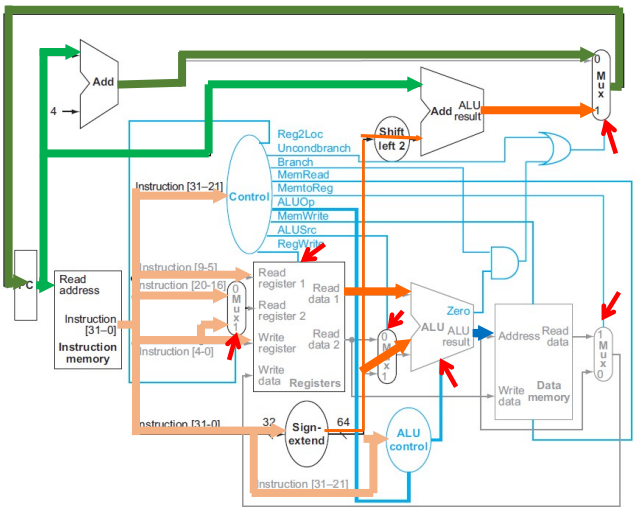


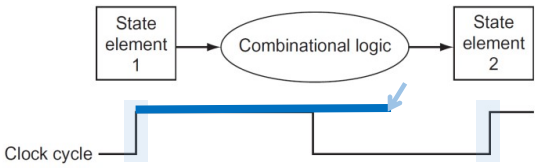


1. Fetch of the instruction and increment PC
2. Read registers and decode the instruction
3. The ALU computes the sum of the value read from the register file and the sign-extended 9 bits of the instruction (offset).

59

LDUR X1, [X2,offset]

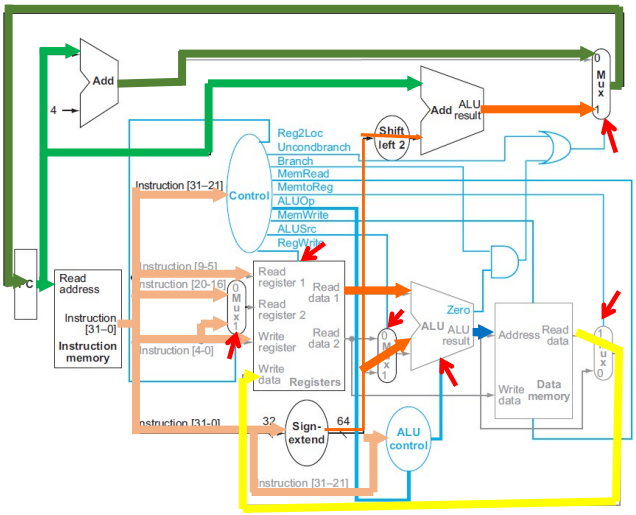


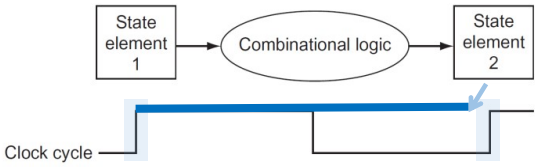


1. Fetch of the instruction and increment PC
2. Read registers and decode the instruction
3. The ALU computes the sum of the value read from the register file and the sign-extended 9 bits of the instruction (offset).
4. The sum from the ALU is used as the address for the data memory.

60

LDUR X1, [X2,offset]

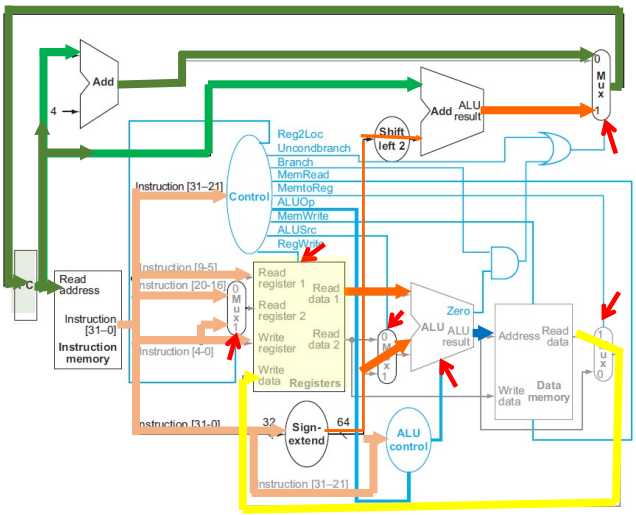


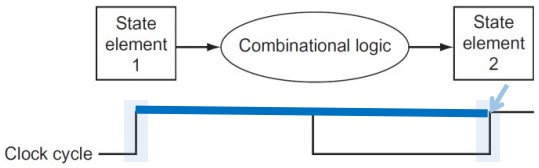


1. Fetch of the instruction and increment PC
2. Read registers and decode the instruction
3. The ALU computes the sum of the value read from the register file and the sign-extended 9 bits of the instruction (offset).
4. The sum from the ALU is used as the address for the data memory.

61

LDUR X1, [X2,offset]





1. Fetch of the instruction and increment PC
2. Read registers and decode the instruction
3. The ALU computes the sum of the value read from the register file and the sign-extended 9 bits of the instruction (offset).
4. The sum from the ALU is used as the address for the data memory.
5. The data from the memory unit is written into the register file (X1).

The state elements all have the clock as an implicit input and that the clock is used in controlling writes.

62

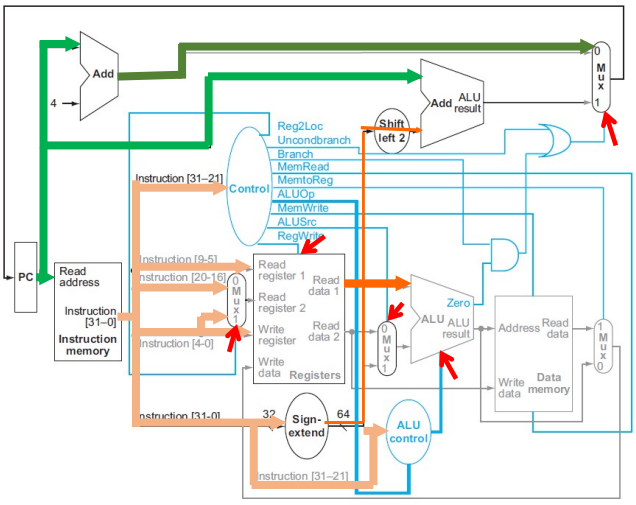
CBZ X1, offset

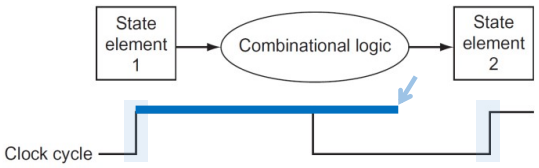
The four steps in execution:

1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. The register, X1 is read from the register file using bits 4:0 of the instruction (Rt).
3. The ALU passes the data value read from the register file. The value of PC is added to the sign-extended, 19 bits of the instruction (offset) are shifted left by two; the result is the branch target address.
4. The Zero status information from the ALU is used to decide which adder result to store in the PC.

63

CBZ X1, offset

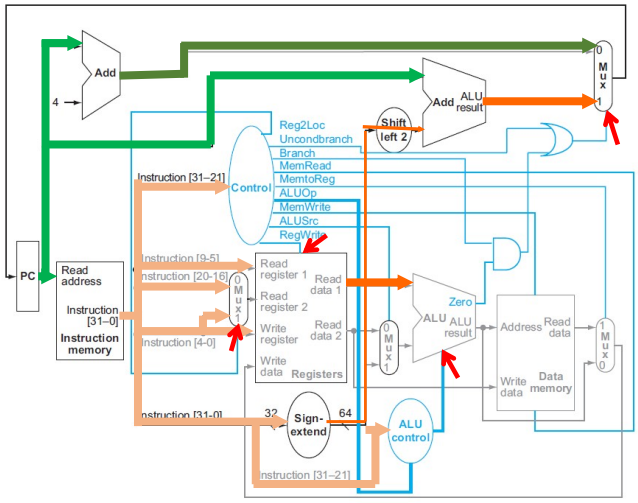


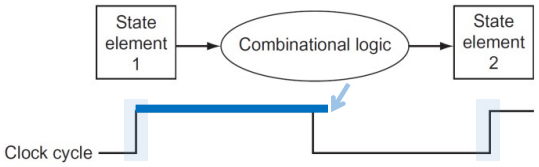


1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. The register, X1 is read from the register file using bits 4:0 of the instruction (Rt).

64

CBZ X1, offset



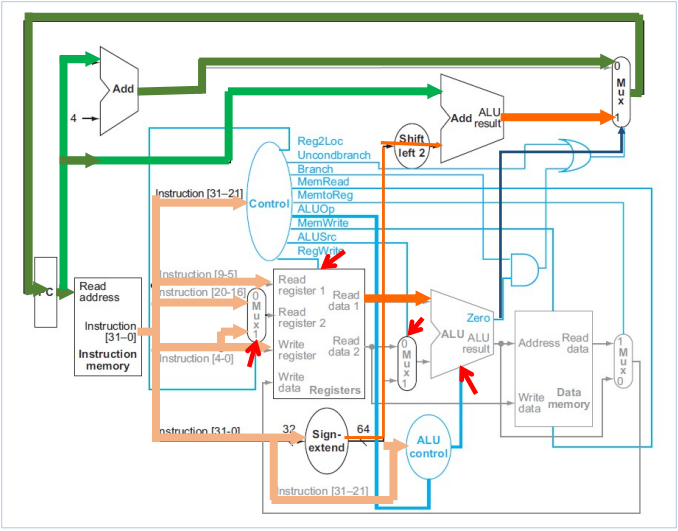


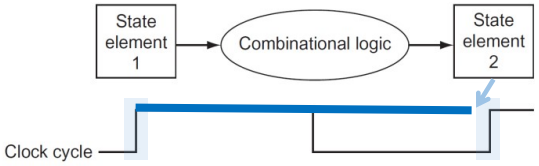
1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. The register, X1 is read from the register file using bits 4:0 of the instruction (Rt).
3. The ALU passes the data value read from the register file. The value of PC is added to the sign-extended, 19 bits of the instruction (offset) are shifted left by two; the result is the branch target address.

65

CBZ X1, offset

No jump
X1 <>0





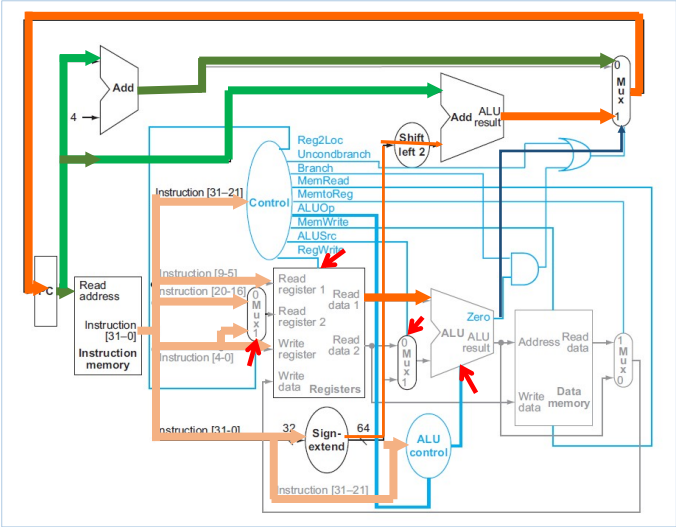
1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. The register, X1 is read from the register file using bits 4:0 of the instruction (Rt).
3. The ALU passes the data value read from the register file. The value of PC is added to the sign-extended, 19 bits of the instruction (offset) are shifted left by two; the result is the branch target address.
4. The Zero status information from the ALU is used to decide which adder result to store in the PC.

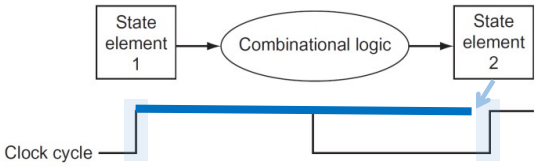
The state elements all have the clock as an implicit input and that the clock is used in controlling writes.

66

CBZ X1, offset

Jump
X1 = 0





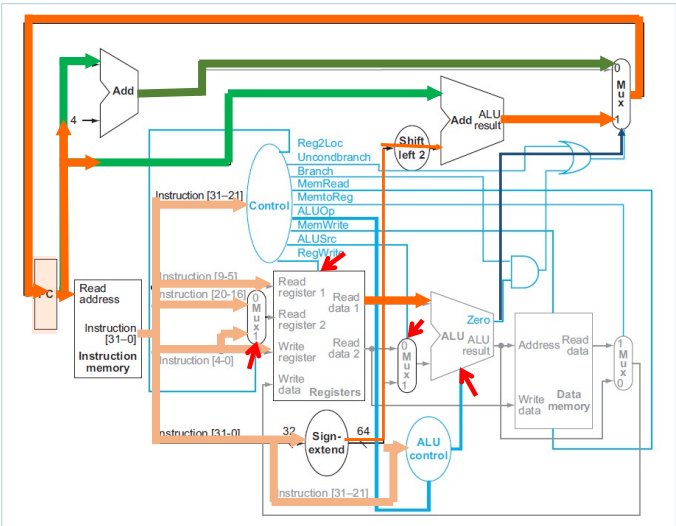
1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. The register, X1 is read from the register file using bits 4:0 of the instruction (Rt).
3. The ALU passes the data value read from the register file. The value of PC is added to the sign-extended, 19 bits of the instruction (offset) are shifted left by two; the result is the branch target address.
4. The Zero status information from the ALU is used to decide which adder result to store in the PC.

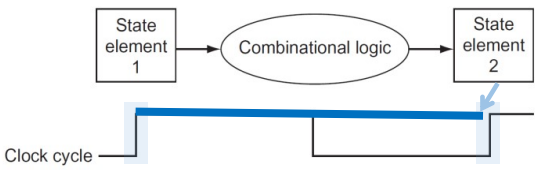
The state elements all have the clock as an implicit input and that the clock is used in controlling writes.

67

CBZ X1, offset

Jump
X1 = 0





1. An instruction is fetched from the instruction memory, and the PC is incremented.
2. The register, X1 is read from the register file using bits 4:0 of the instruction (Rt).
3. The ALU passes the data value read from the register file. The value of PC is added to the sign-extended, 19 bits of the instruction (offset) are shifted left by two; the result is the branch target address.
4. The Zero status information from the ALU is used to decide which adder result to store in the PC.

The state elements all have the clock as an implicit input and that the clock is used in controlling writes.

68