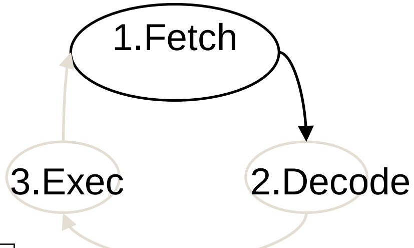
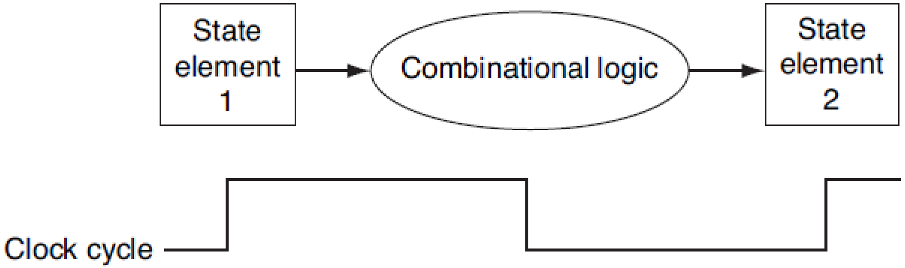
CHAPTER 5-2

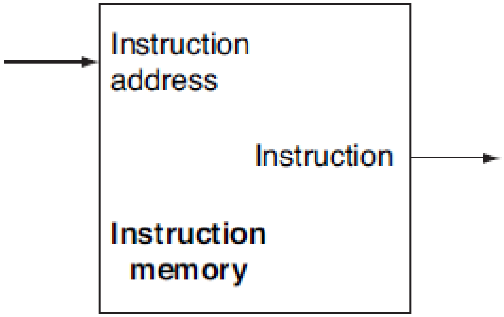
The Processor

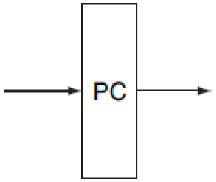
By Pattama Longani
Collage of arts, media and Technology

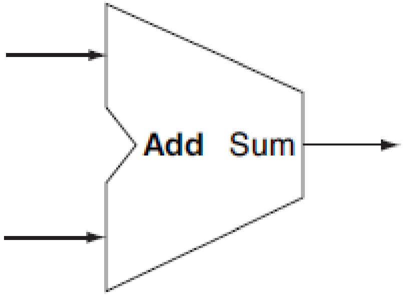
DATAPATHS

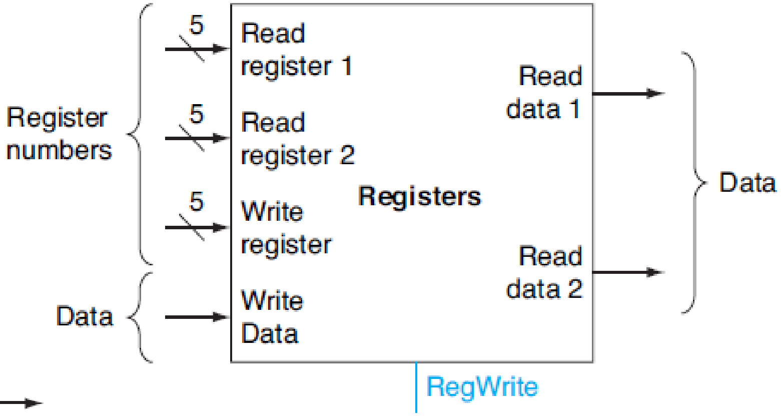

1.Fetch
2.Decode
3.Exec

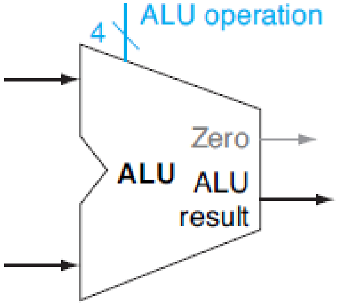

State element 1
Combinational logic
State element 2
Clock cycle

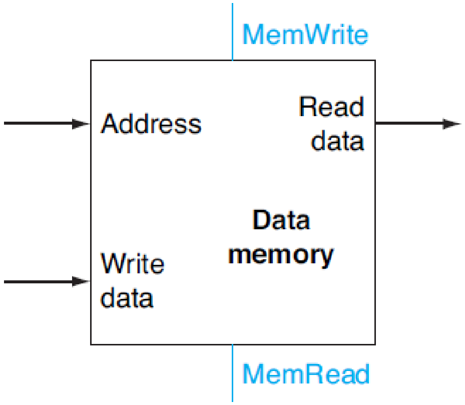

Instruction address
Instruction
Instruction memory

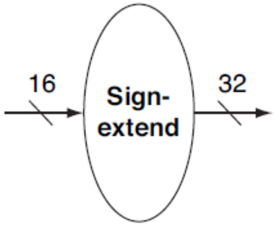

PC


Add
Sum


Register numbers
5
Read register 1
5
Read register 2
5
Write register
Data
Write Data
Registers
Read data 1
Read data 2
Data
RegWrite

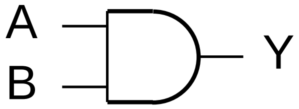

4
ALU operation
Zero
ALU
ALU result


MemWrite
Address
Read data
Data memory
Write data
MemRead

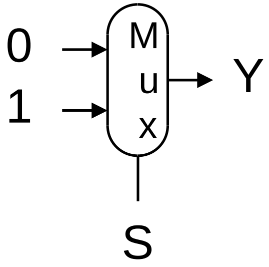

16
Sign-extend
32

COMBINATIONAL ELEMENTS

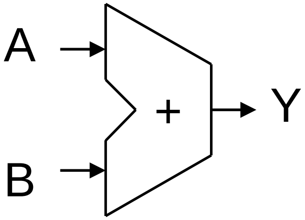
- AND-gate
  - Y = A & B



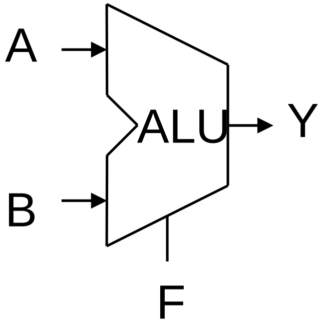
- Multiplexer
  - Y = 0 ? 0 : 1



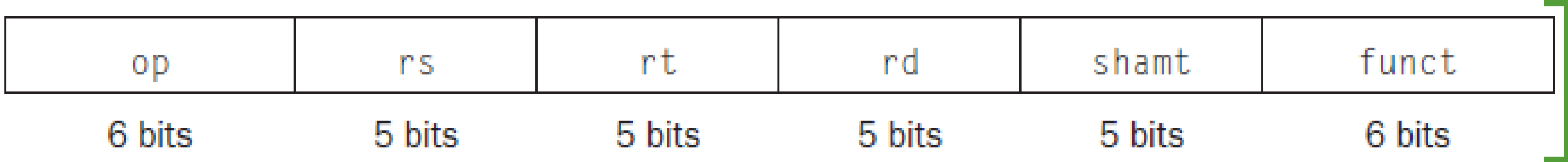
- Adder
  - Y = A + B



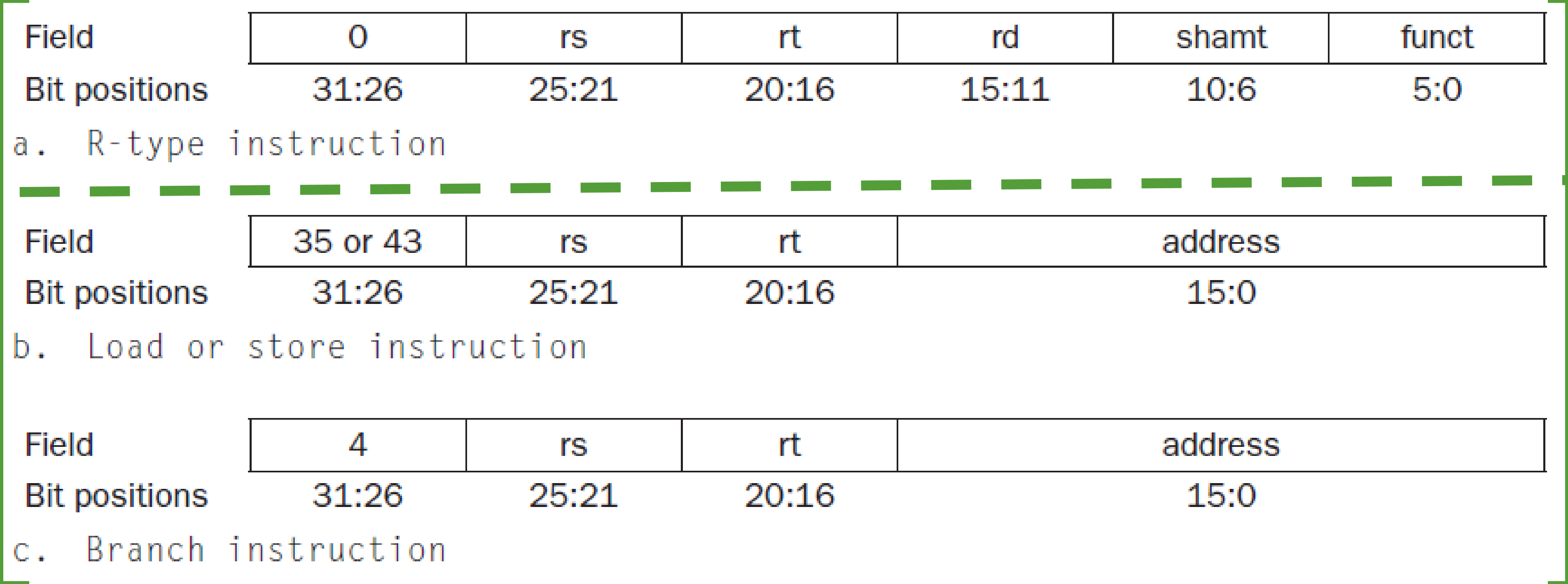
- Arithmetic/Logic Unit
  - Y = F(A, B)



In this section, we use **MIPS subset** to build simple implementation using the datapath of the last section and adding a simple control function



op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



Field	0	rs	rt	rd	shamt	funct
Bit positions	31:26	25:21	20:16	15:11	10:6	5:0

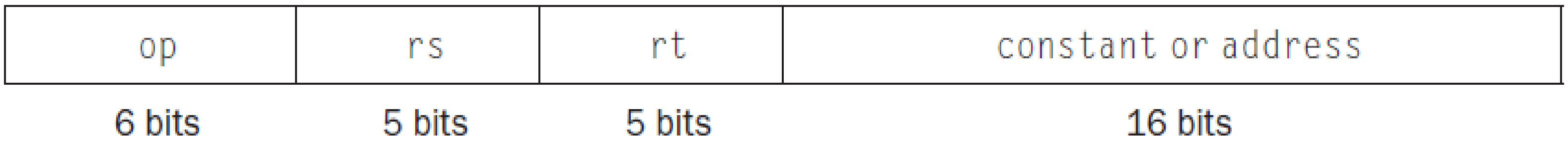
a. R-type instruction

Field	35 or 43	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

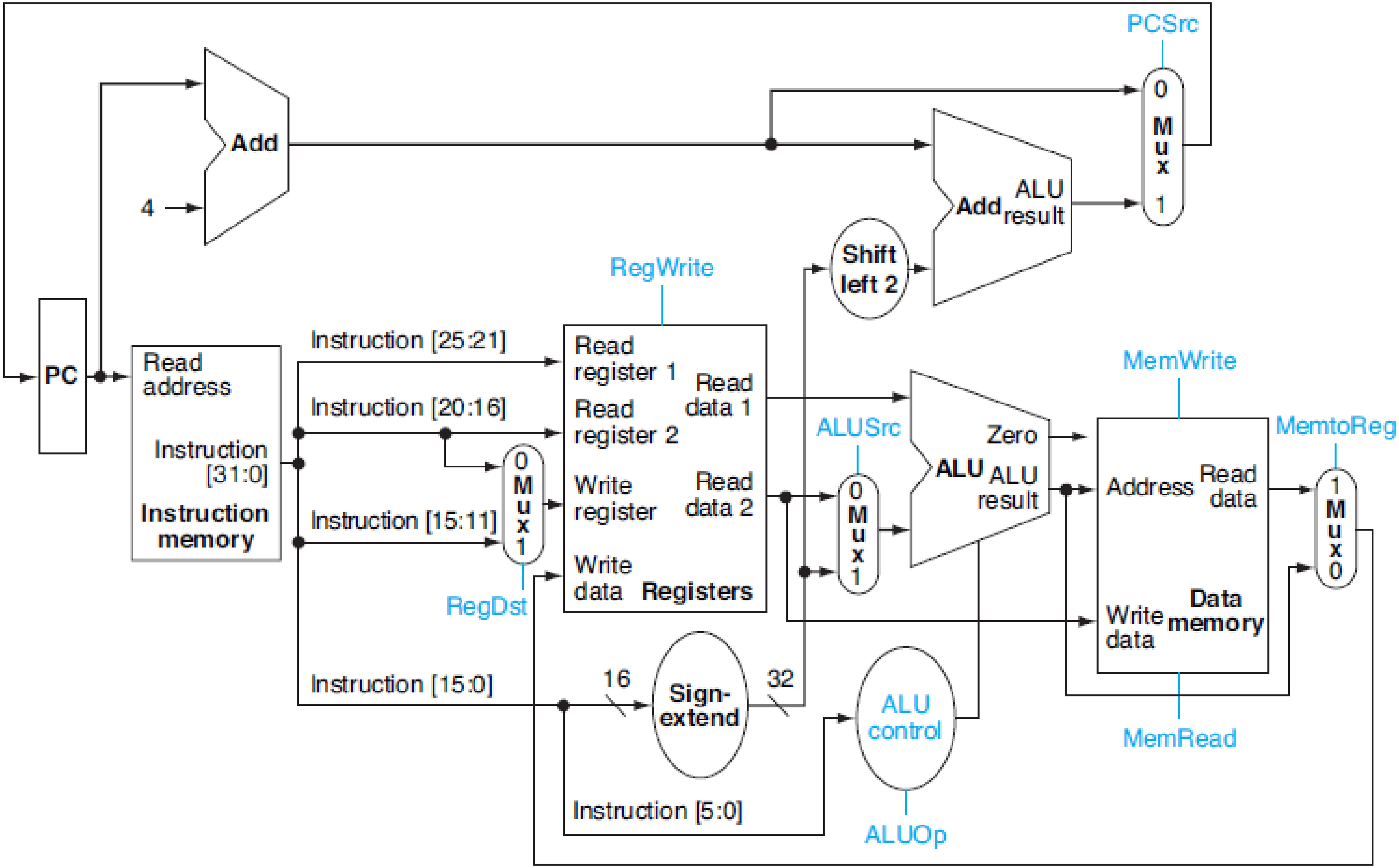
b. Load or store instruction

Field	4	rs	rt	address
Bit positions	31:26	25:21	20:16	15:0

c. Branch instruction

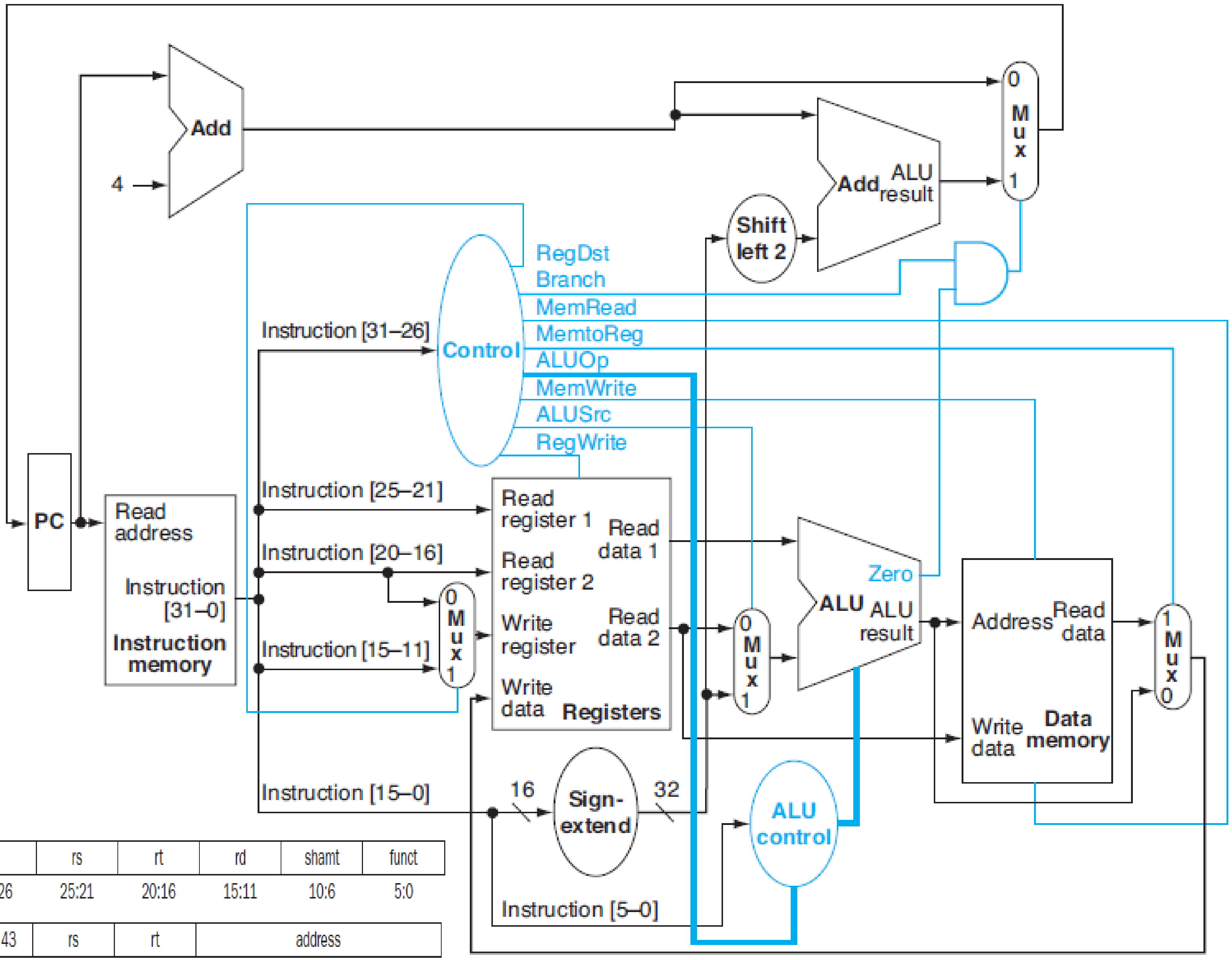


op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits



0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

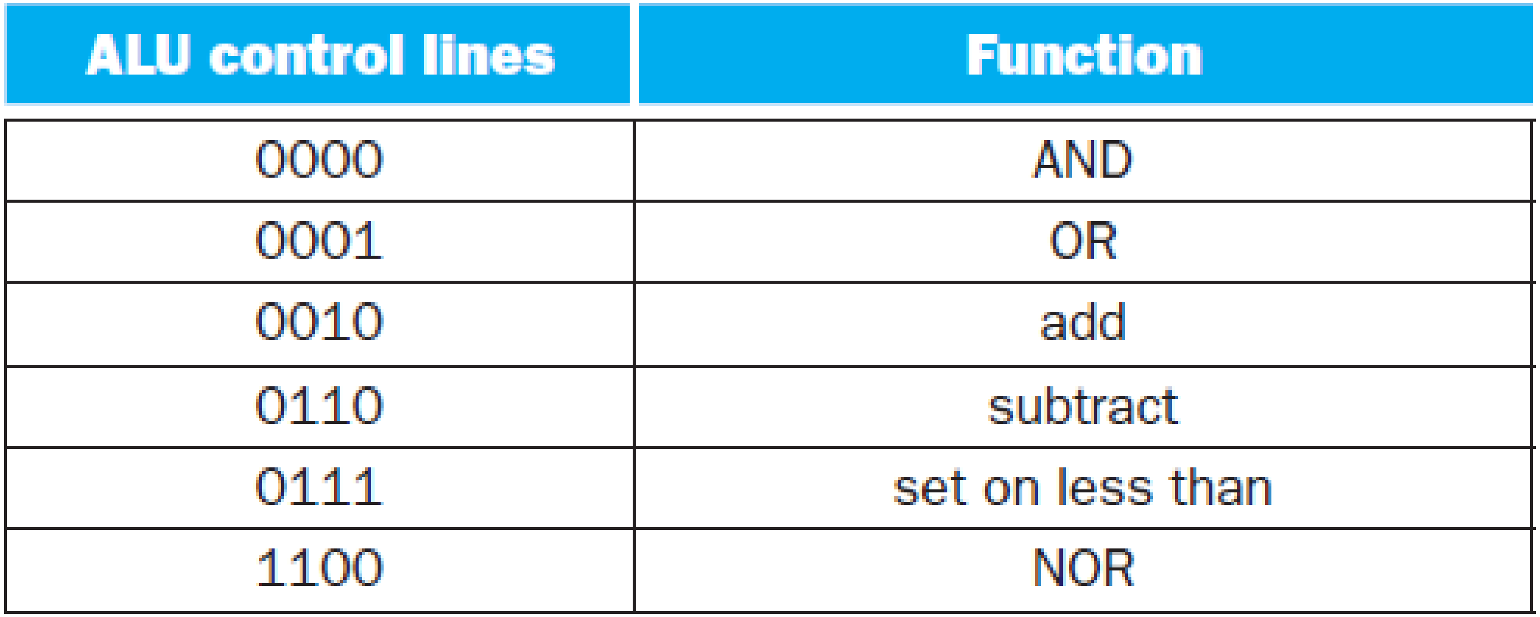
35 or 43	rs	rt	address
31:26	25:21	20:16	15:0



0	rs	rt	rd	shamt	funct
31:26	25:21	20:16	15:11	10:6	5:0

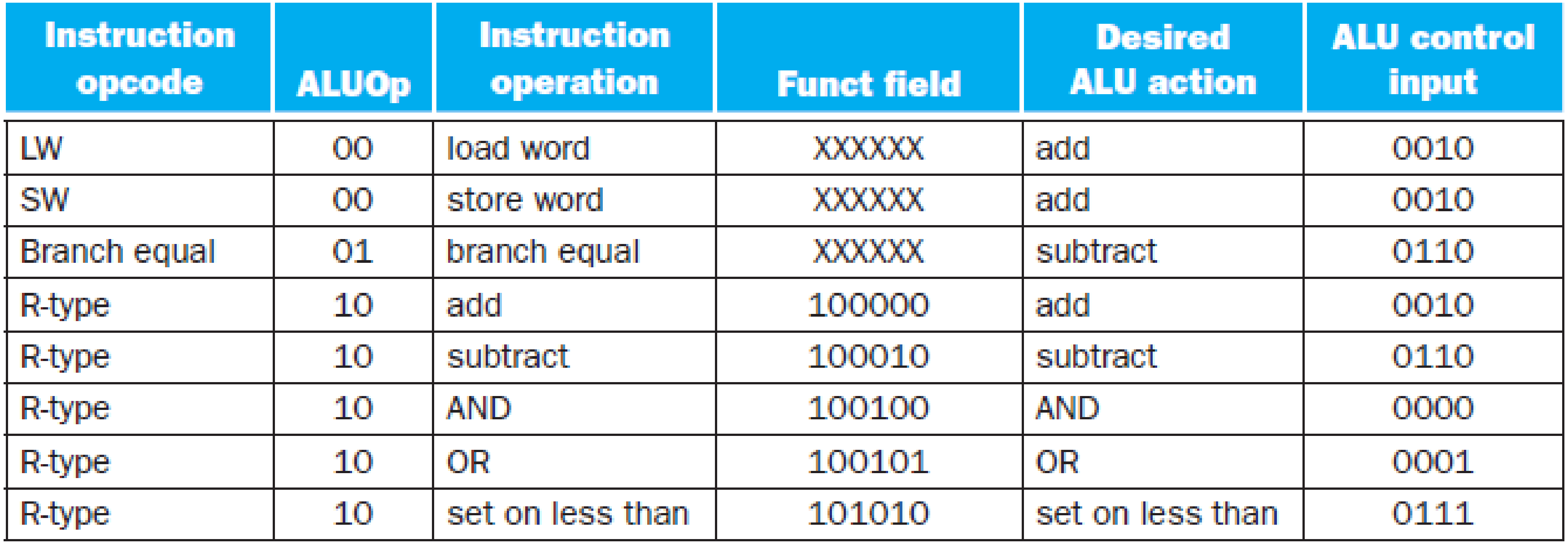
35 or 43	rs	rt	address
31:26	25:21	20:16	15:0

FOUR CONTROL INPUTS OF ALU



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set on less than
1100	NOR

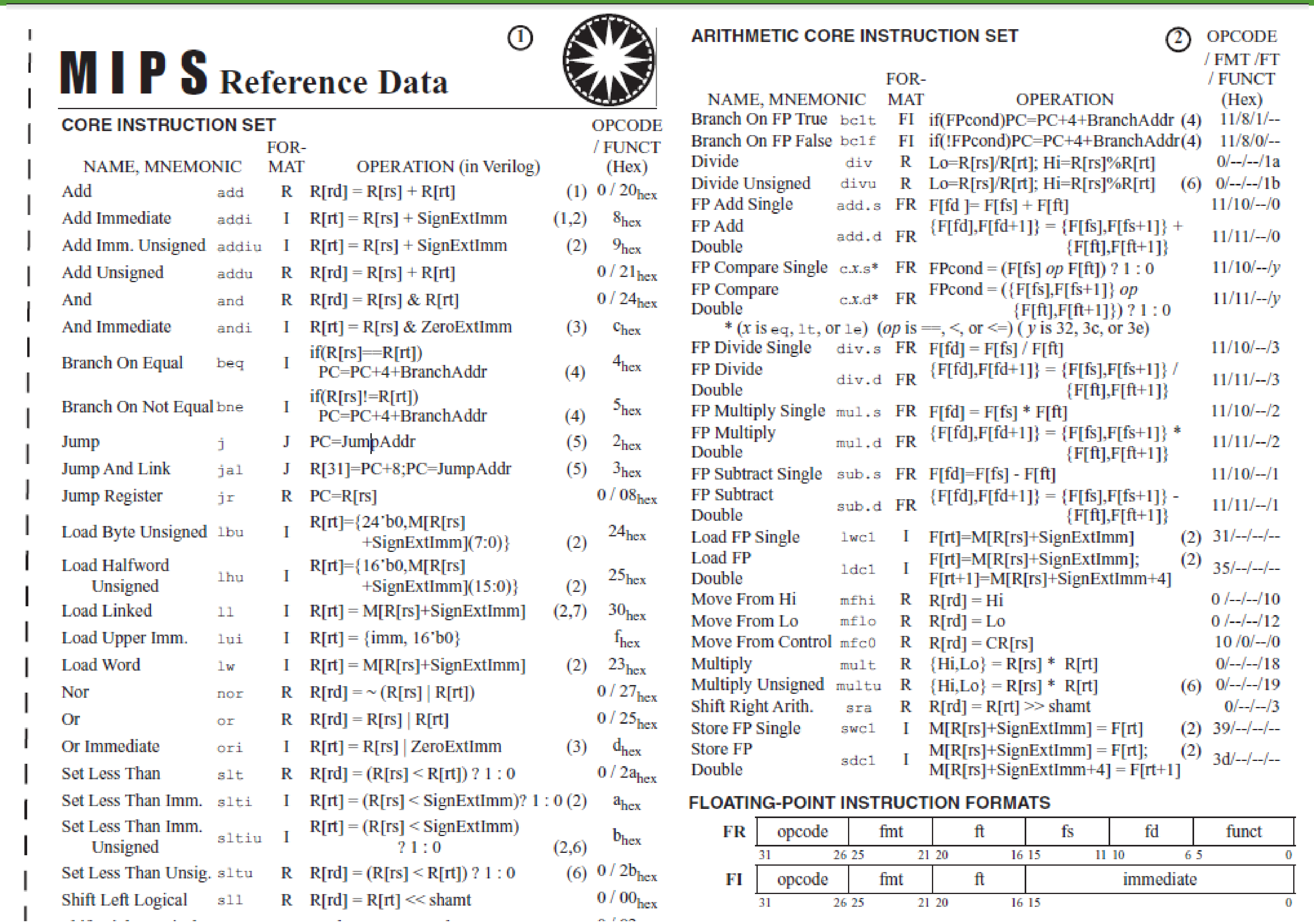
- For load word and store word instructions, we use the ALU to compute the memory address by addition



Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

ALUOp indicates whether the operation to be performed should be

- add (00) : for loads and stores,
- subtract (01) : for beq,
- determined by the operation
- encoded in the funct field (10) : R-type instruction



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_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}

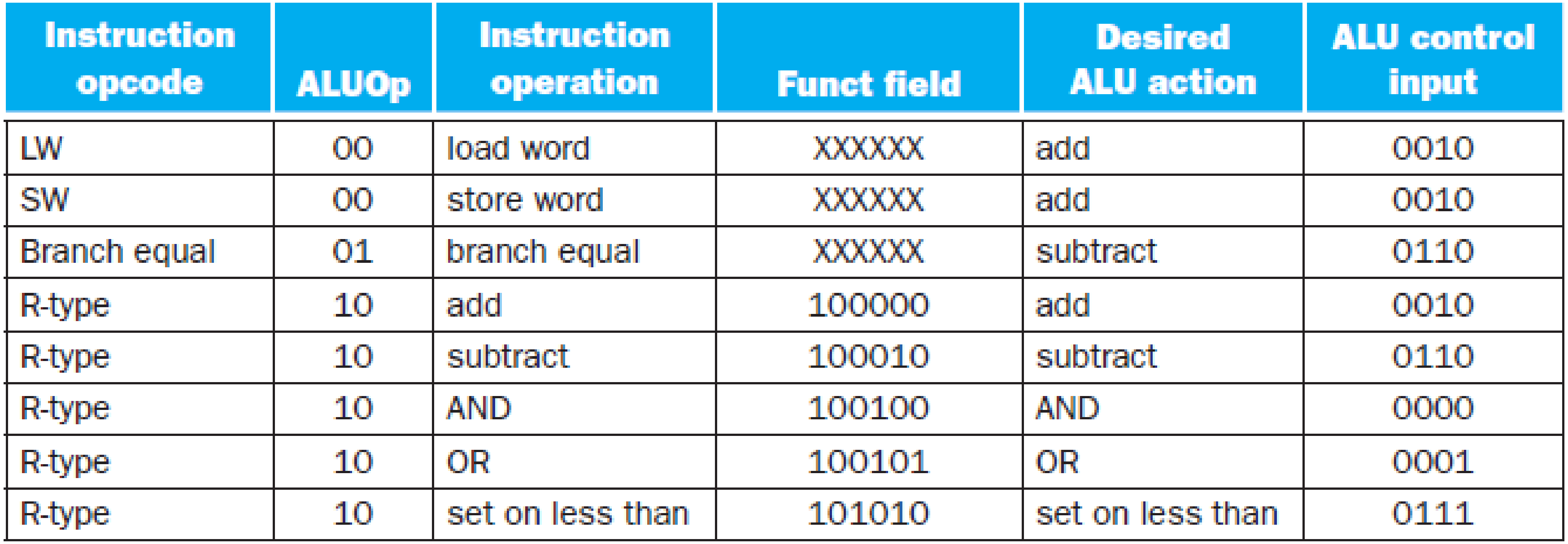
ARITHMETIC CORE INSTRUCTION SET ②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+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] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.*x*.d*	FR	FPcond = ({F[fs],F[fs+1]} *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]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+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] >> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS

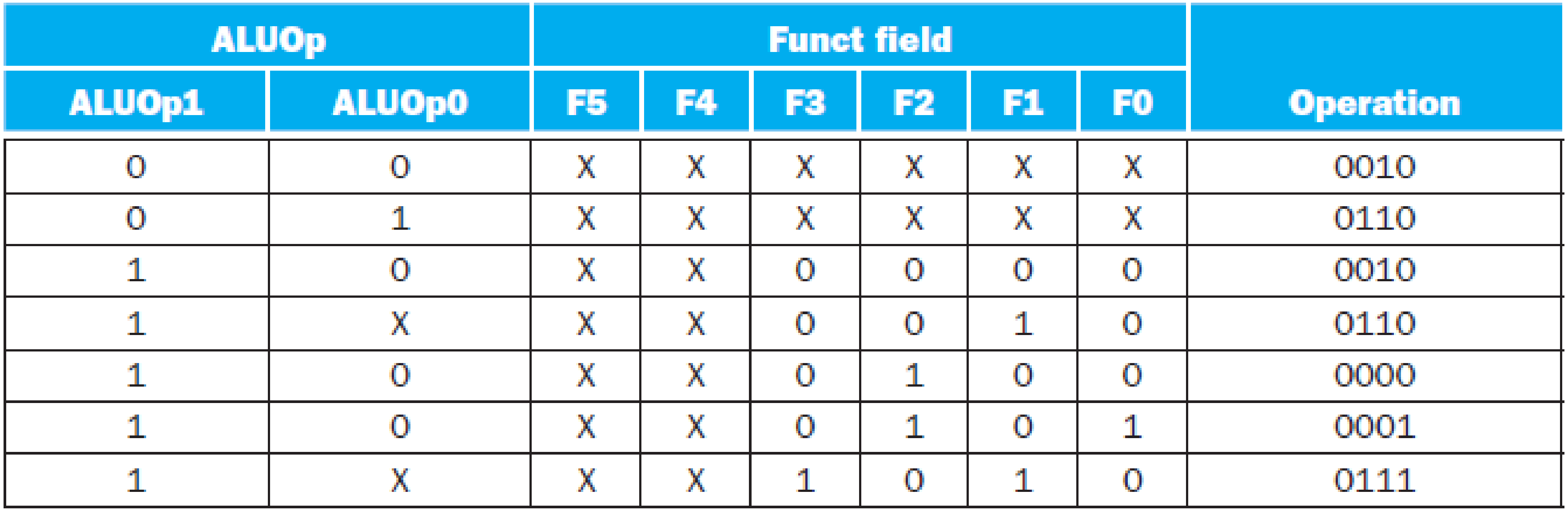
FR	opcode	fmt	ft	fs	fd	funct
	31 26	25 21	20 16	15 11	10 6	5 0
FI	opcode	fmt	ft	immediate		
	31 26	25 21	20 16	15 0		

- Note that: R-type instructions works according to the function code part.



Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

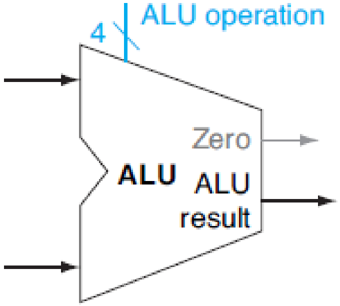
X= don’t care term

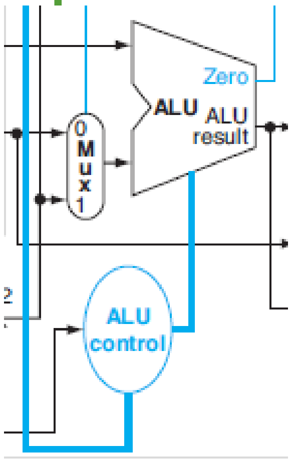


ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

ALU CONTROL

- **ALUOp** indicates whether the operation to be performed should be
  - 00 = add
  - 01 = sub
  - 10 = function field

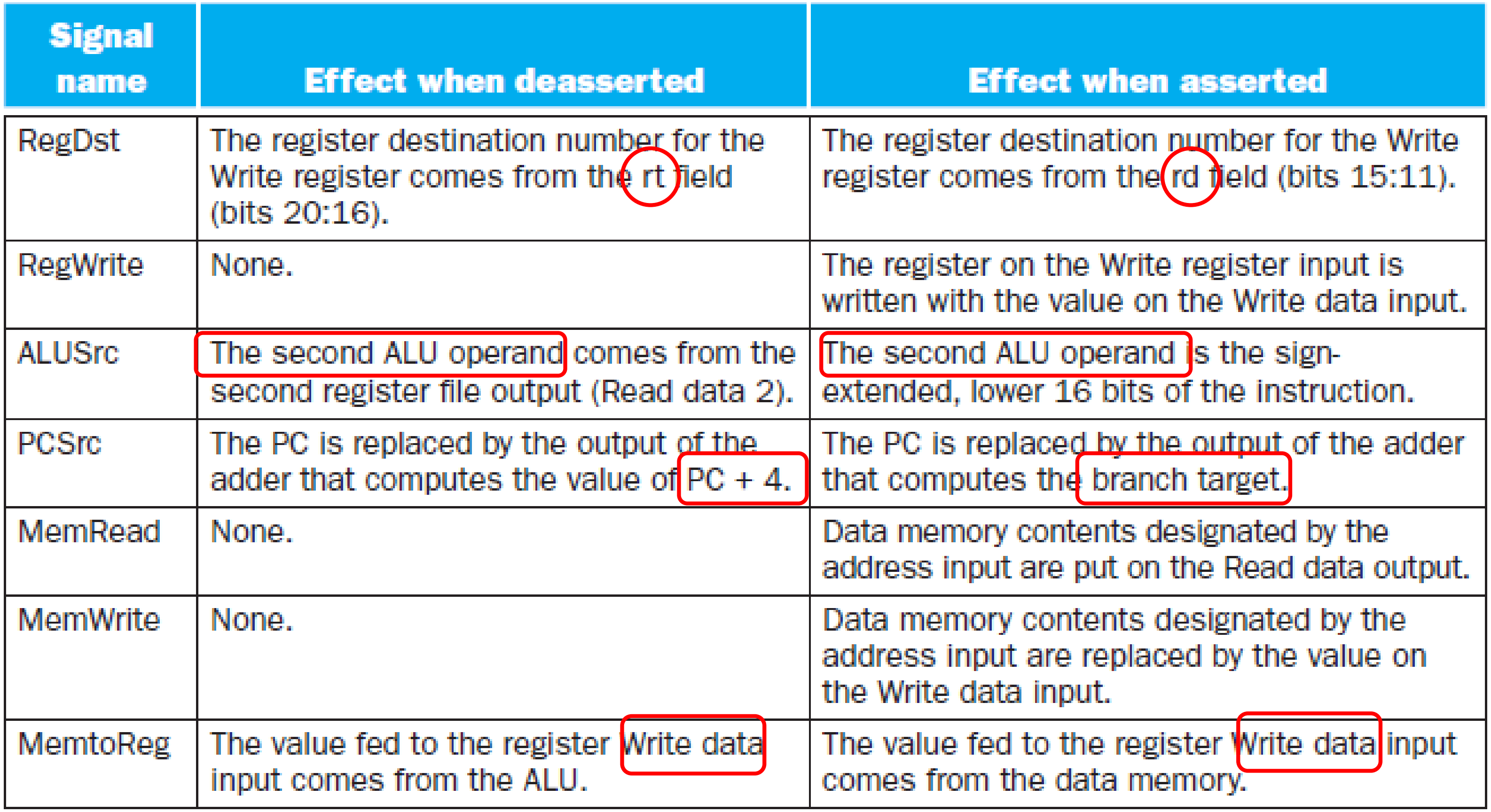




ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
0	1	X	X	X	X	X	X	0110
1	0	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	0	X	X	0	1	0	0	0000
1	0	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

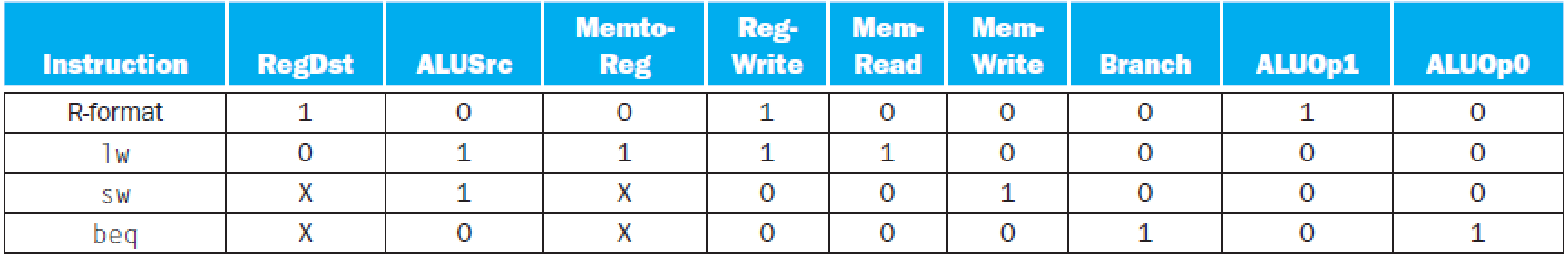
0

1

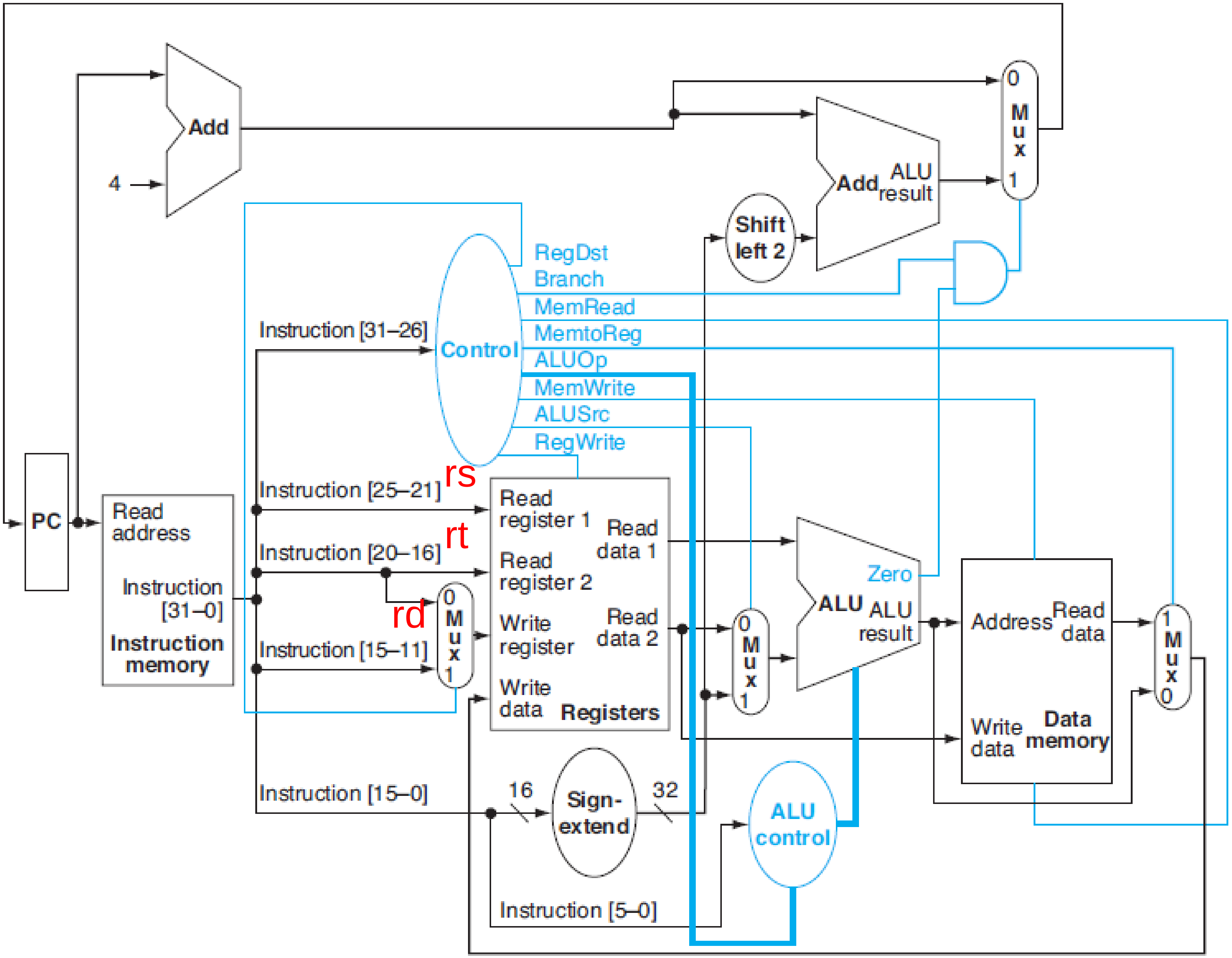


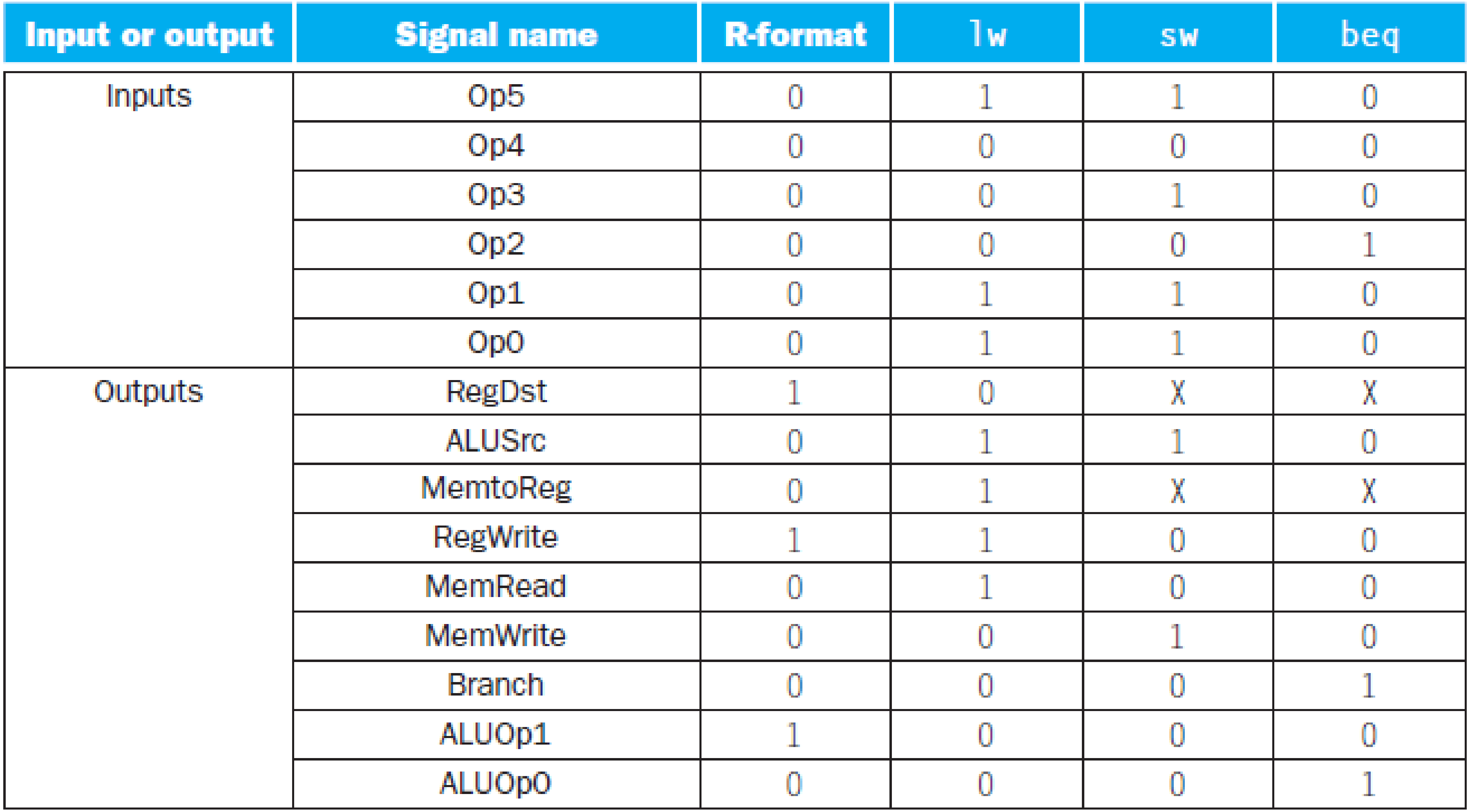
Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input 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.

Control



Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1


Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
rs
Instruction [20–16]
rt
rd
Instruction [15–11]
0
M
u
x
1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Instruction [15–0]
16
Sign-extend
32
Instruction [5–0]
ALU control
Shift left 2
Add
ALU result
Zero
ALU
ALU result
Address
Read data
Write data
Data memory



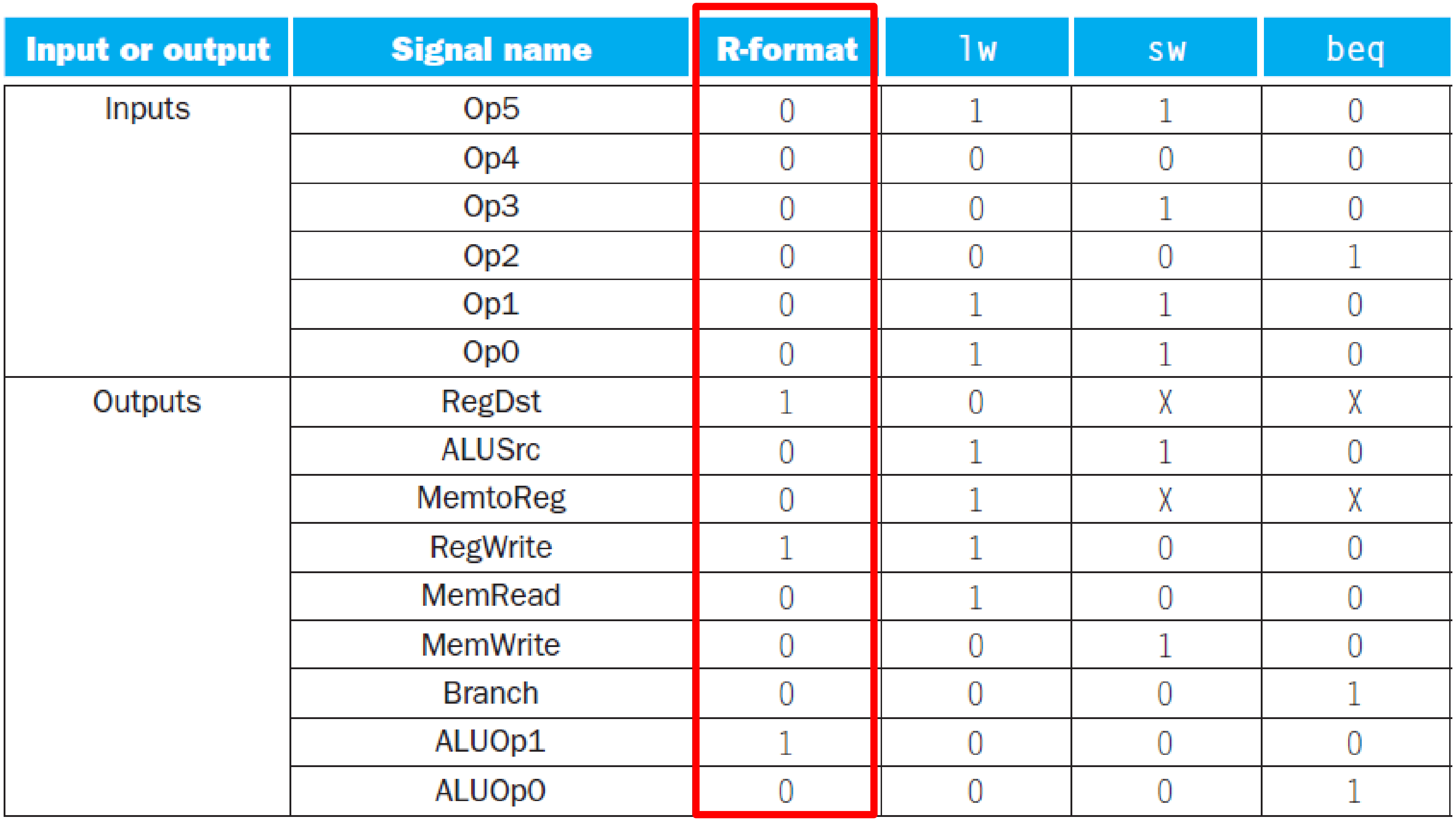
			35_{ten}	43_{ten}	4_{ten}
Input or output	**Signal name**	**R-format**	**lw**	**sw**	**beq**
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

00 = add

01 = sub

10 = function field

ADD = R-TYPE

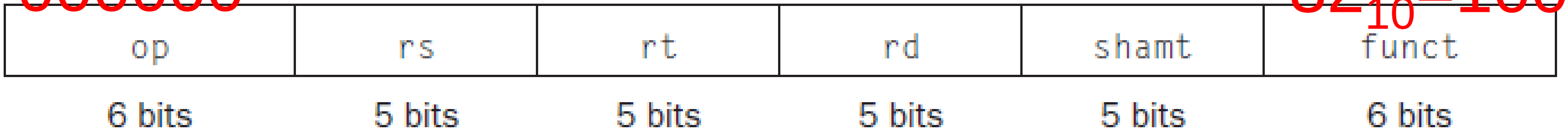


Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALUOp 10 = function field

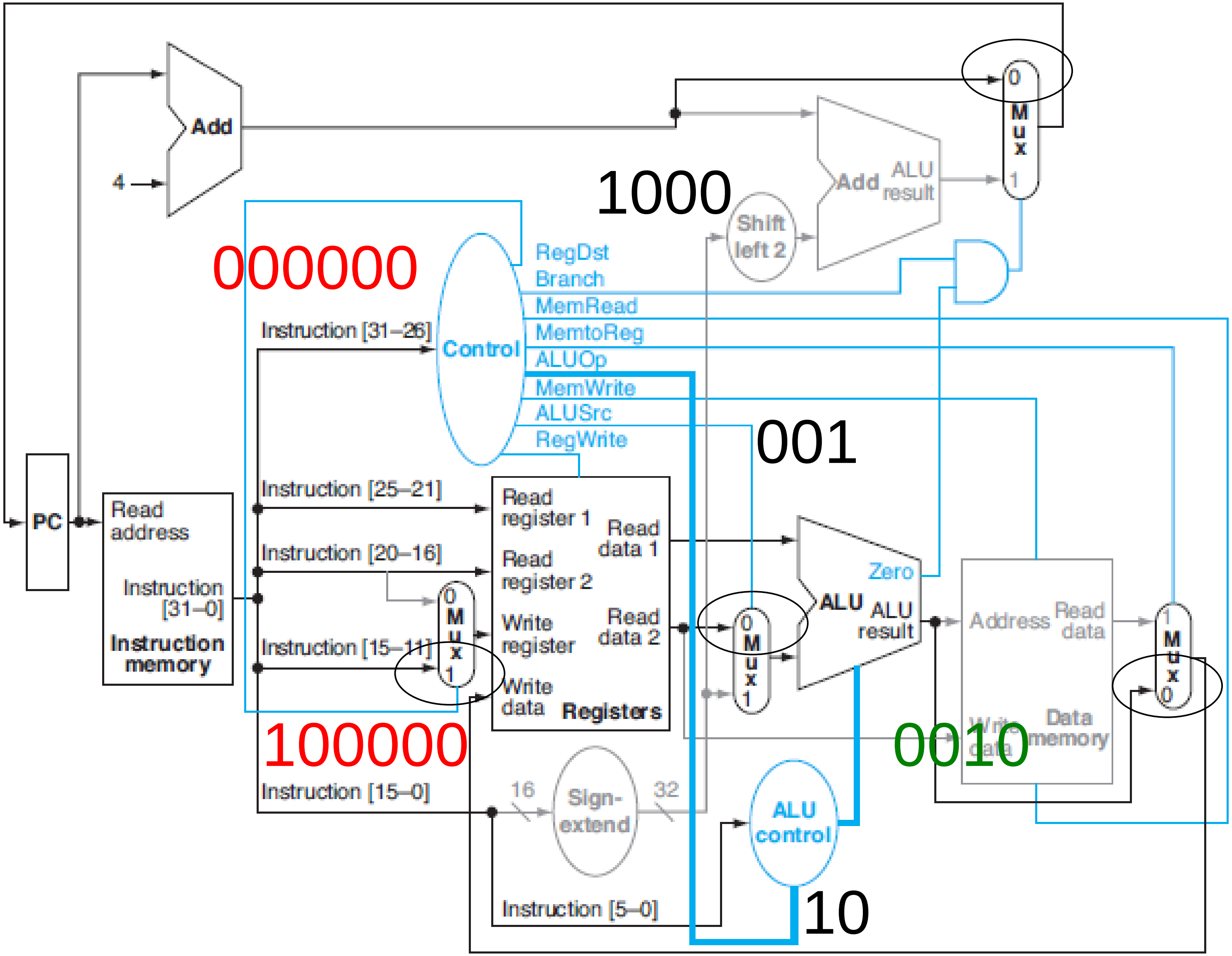
000000

32_{10}=100000



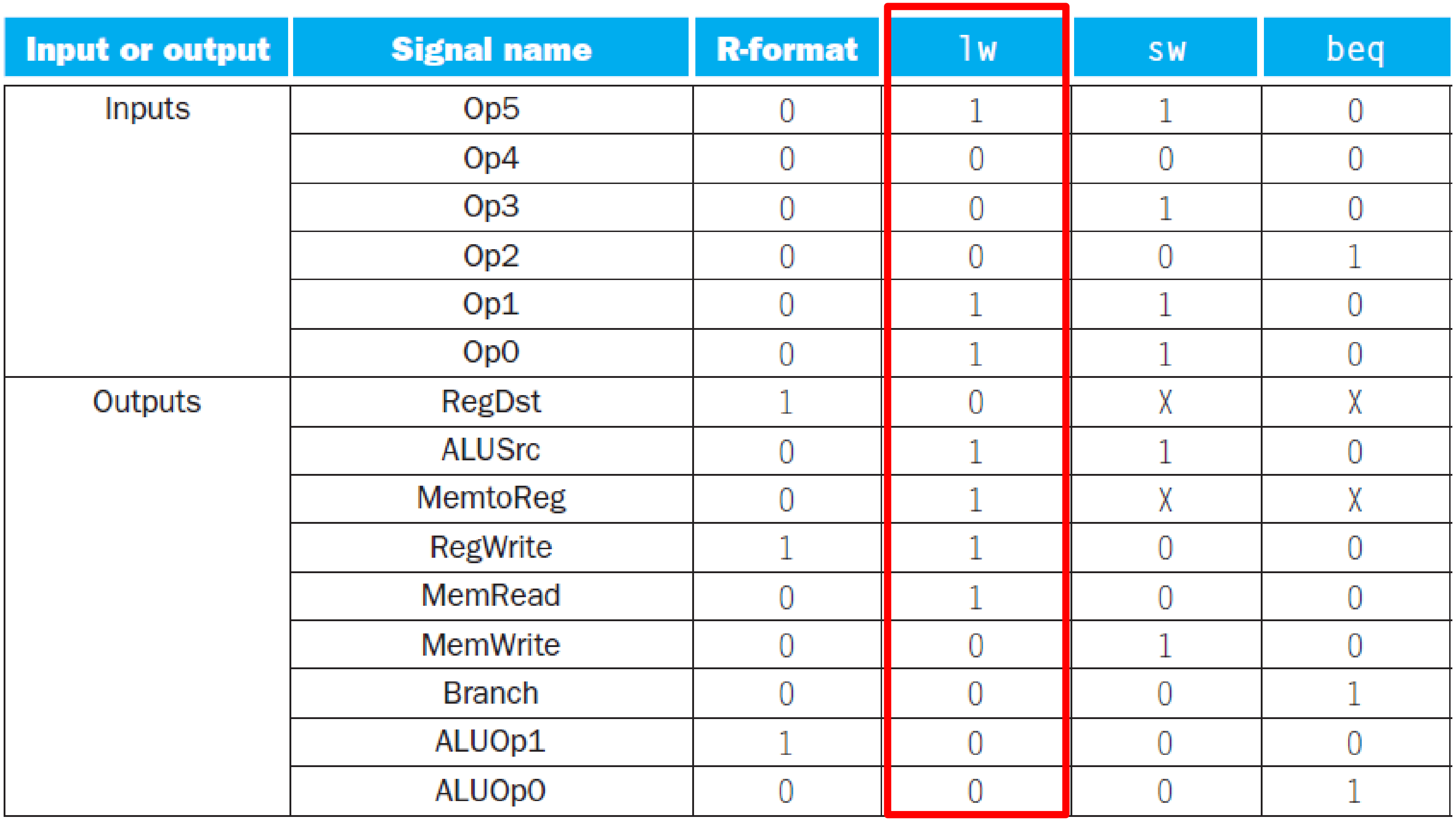
op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

ADD = R-TYPE



Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
000000
1000
001
100000
0010
10
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
Sign-extend
32
Shift left 2
Add
ALU result
0
M
u
x
1
Zero
ALU
ALU result
ALU control
Address
Read data
Write data
Data memory

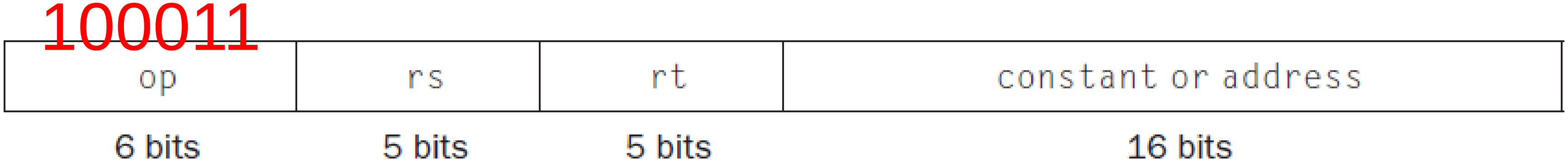
Load = I-Type



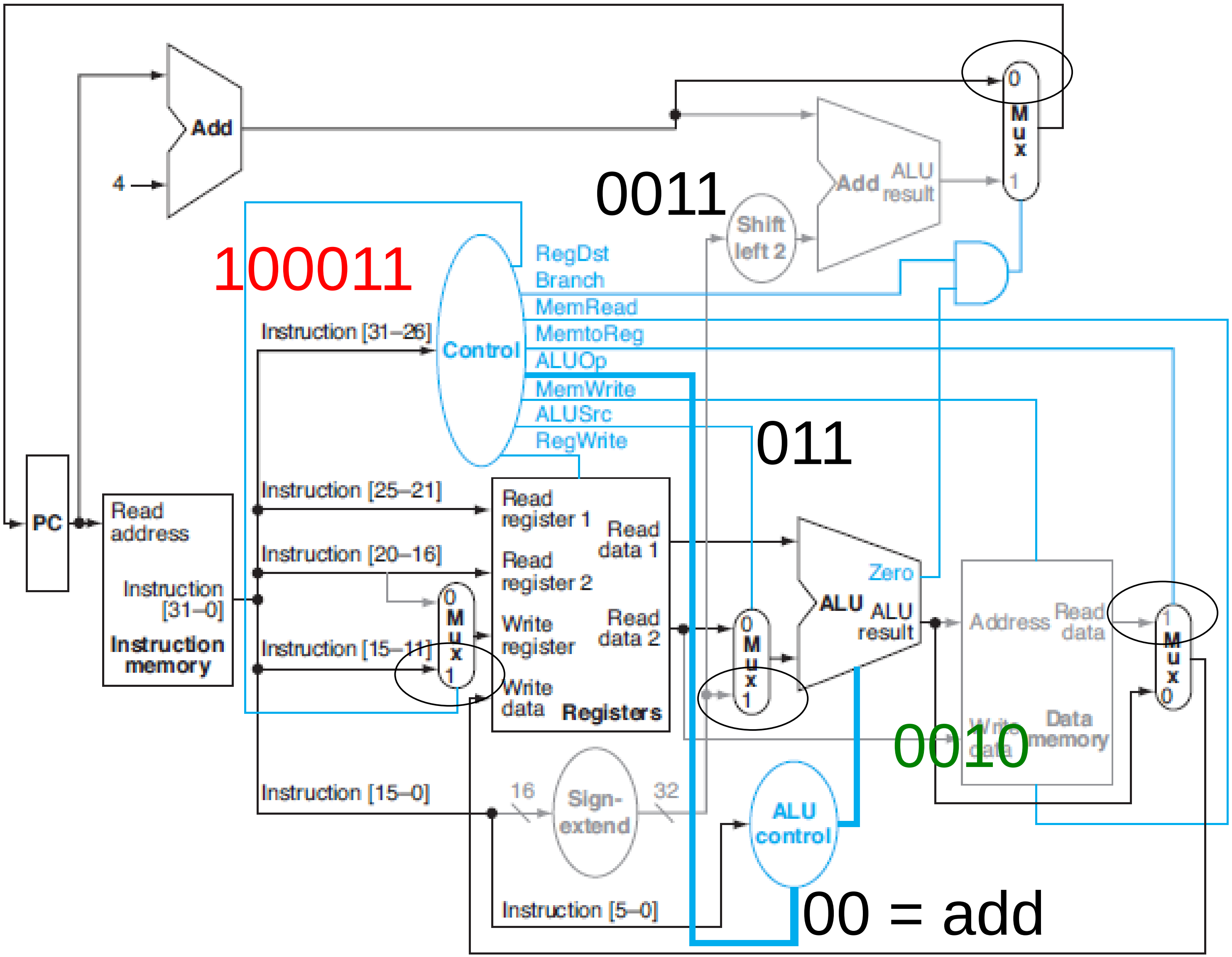
Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

ALUOp 00 = add

100011



op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits



100011

0011

011

0010

00 = add

SINGLE-CYCLE IMPLEMENTATION

- everything occurs in one clock cycle
- There are four steps to execute the instruction; (ordered by the flow of information)

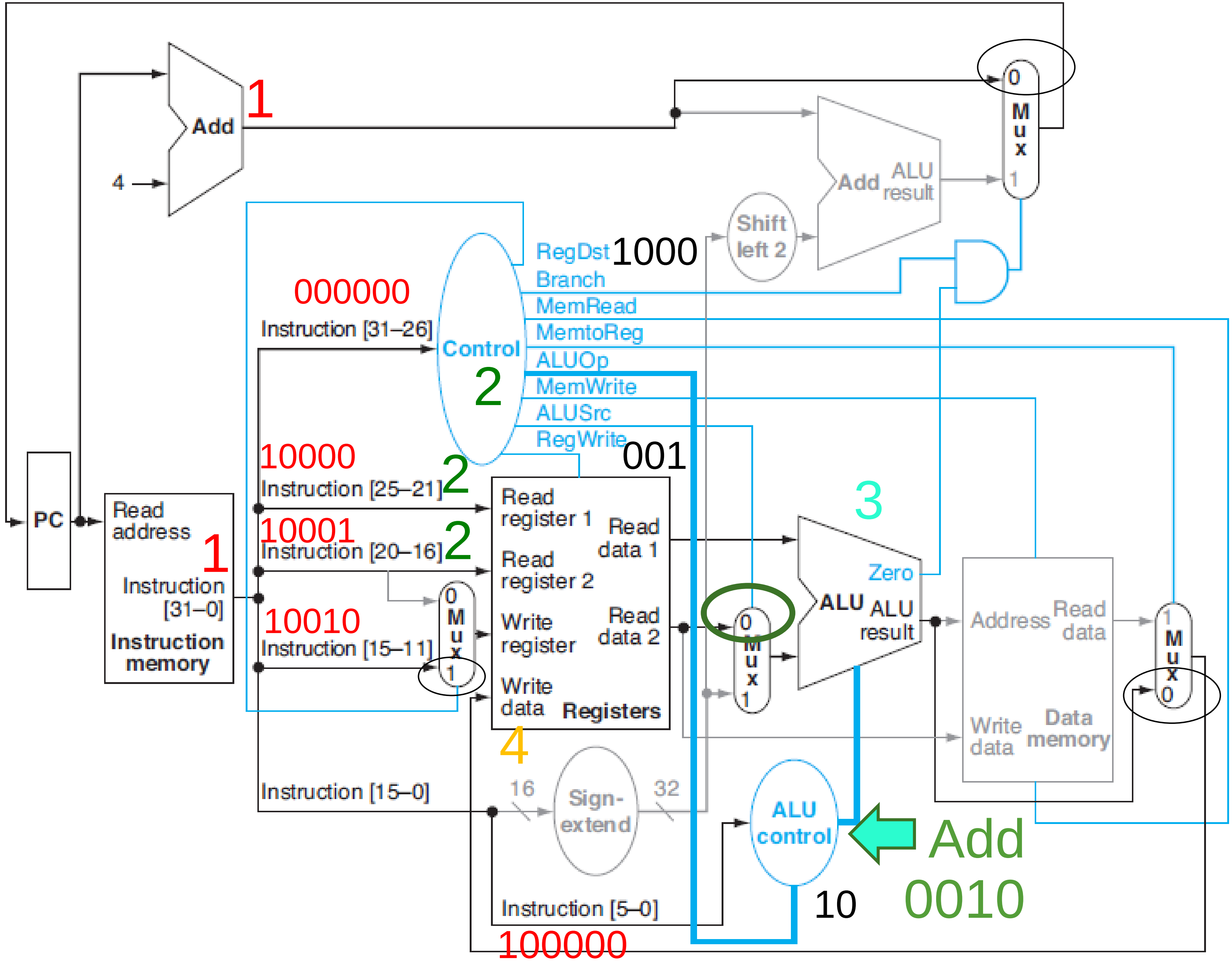
EXAMPLE : add $s1 $s2 $s3

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

2. Two registers, $s2 and $s3, 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 the function code (bits 5:0, which is the funct field, of the instruction) to generate the ALU function.

4. The result from the ALU is written into the register file using bits 15:11 of the instruction to select the destination register ($s1).


Add
4
1
PC
Read address
Instruction [31–0]
Instruction memory
1
000000
Instruction [31–26]
Control
2
RegDst
1000
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
001
10000
Instruction [25–21]
2
10001
Instruction [20–16]
2
10010
Instruction [15–11]
0
M
u
x
1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
4
Instruction [15–0]
16
Sign-extend
32
Instruction [5–0]
100000
Shift left 2
Add
ALU result
0
M
u
x
1
3
0
M
u
x
1
Zero
ALU
ALU result
ALU control
10
Add
0010
Address
Read data
Write data
Data memory
1
M
u
x
0

EXAMPLE : lw $t1, offset($t2)

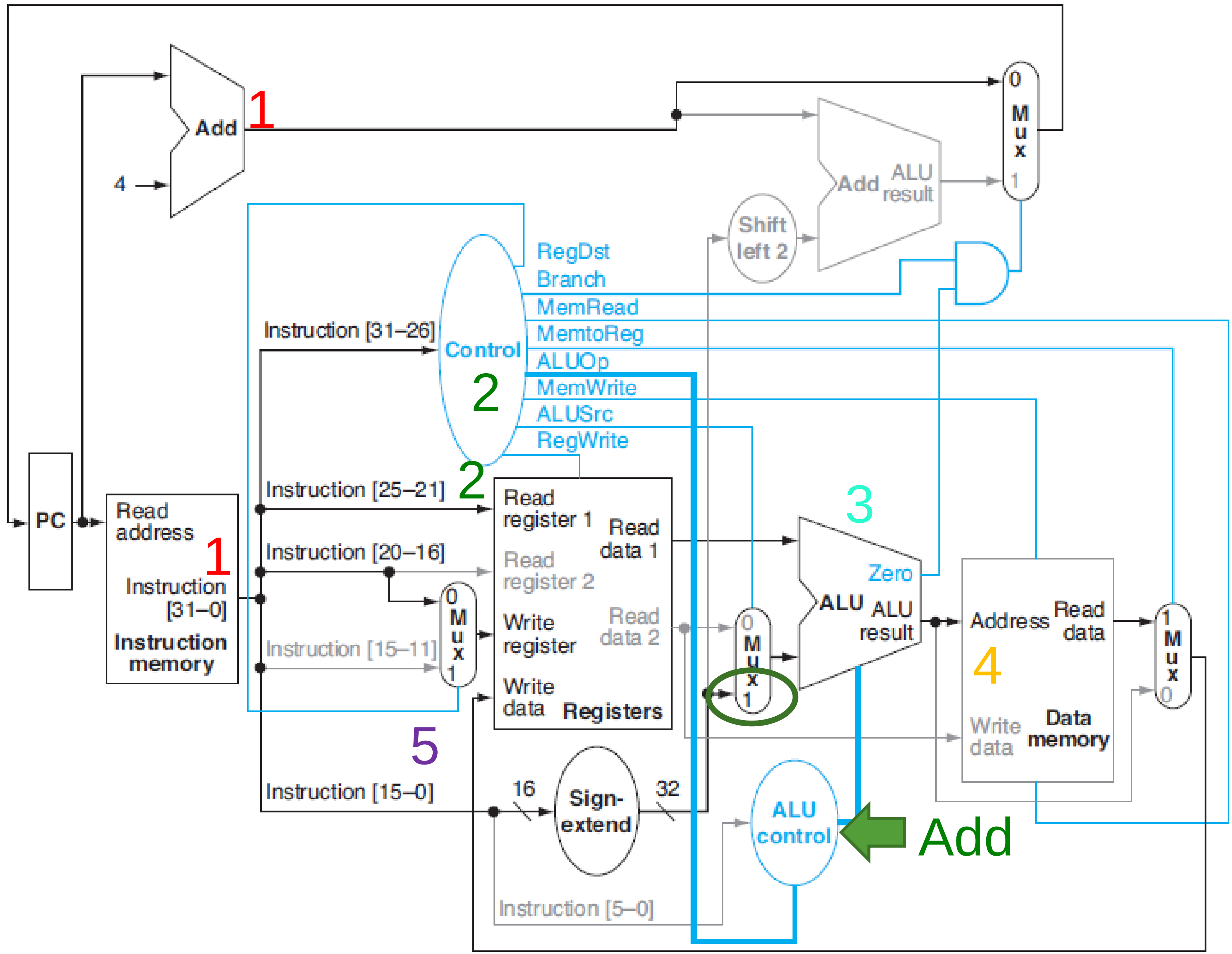
1. An instruction is fetched from the instruction memory, and the PC is incremented.

2. A register ($t2) 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, lower 16 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; the register destination is given by bits 20:16 of the instruction ($t1) .


Add
1
4
PC
Read address
1
Instruction [31–0]
Instruction memory
Instruction [31–26]
Control
2
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [25–21]
2
Instruction [20–16]
Instruction [15–11]
0 M u x 1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
5
Instruction [15–0]
16
Sign-extend
32
Instruction [5–0]
Shift left 2
Add ALU result
0 M u x 1
3
Zero
ALU ALU result
0 M u x 1
ALU control
Add
Address
Read data
4
Write data
Data memory
1 M u x 0

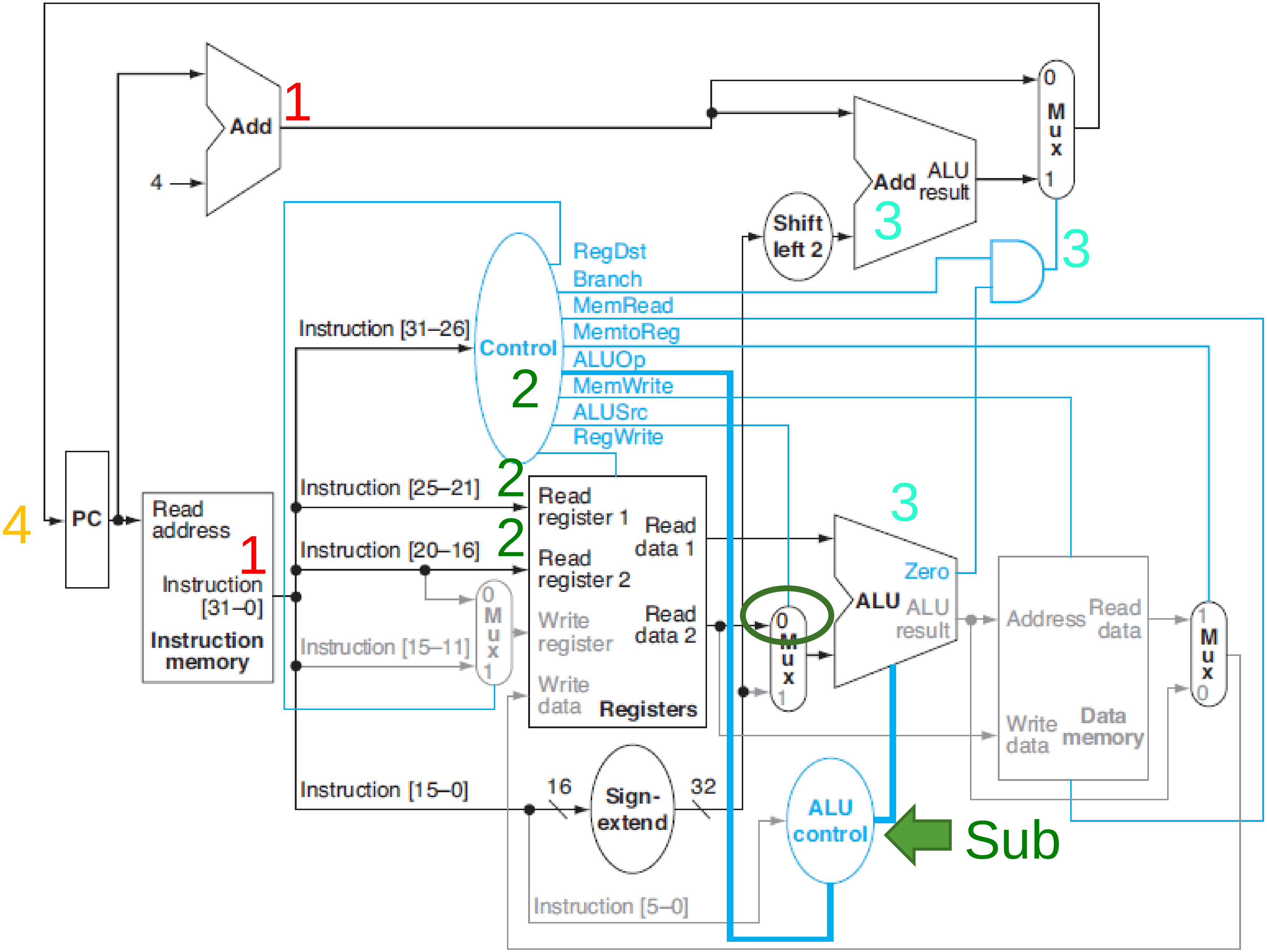
EXAMPLE : beq $t1,$t2,offset,

1. An instruction is fetched from the instruction memory, and the PC is incremented.

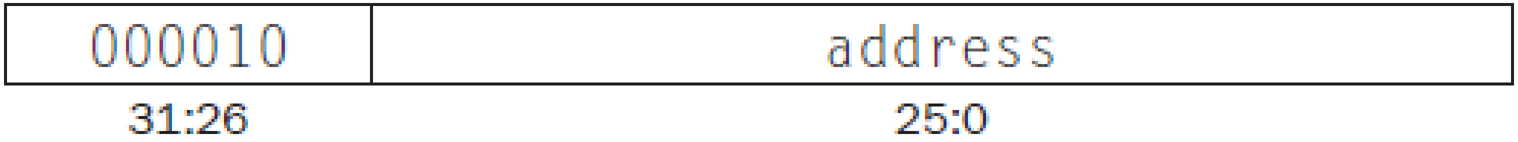
2. Two registers, $t1 and $t2, are read from the register file.

3. The ALU performs a subtract on the data values read from the register file. The value of PC + 4 is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; the result is the branch target address.

4. The Zero result from the ALU is used to decide which adder result to store into the PC.

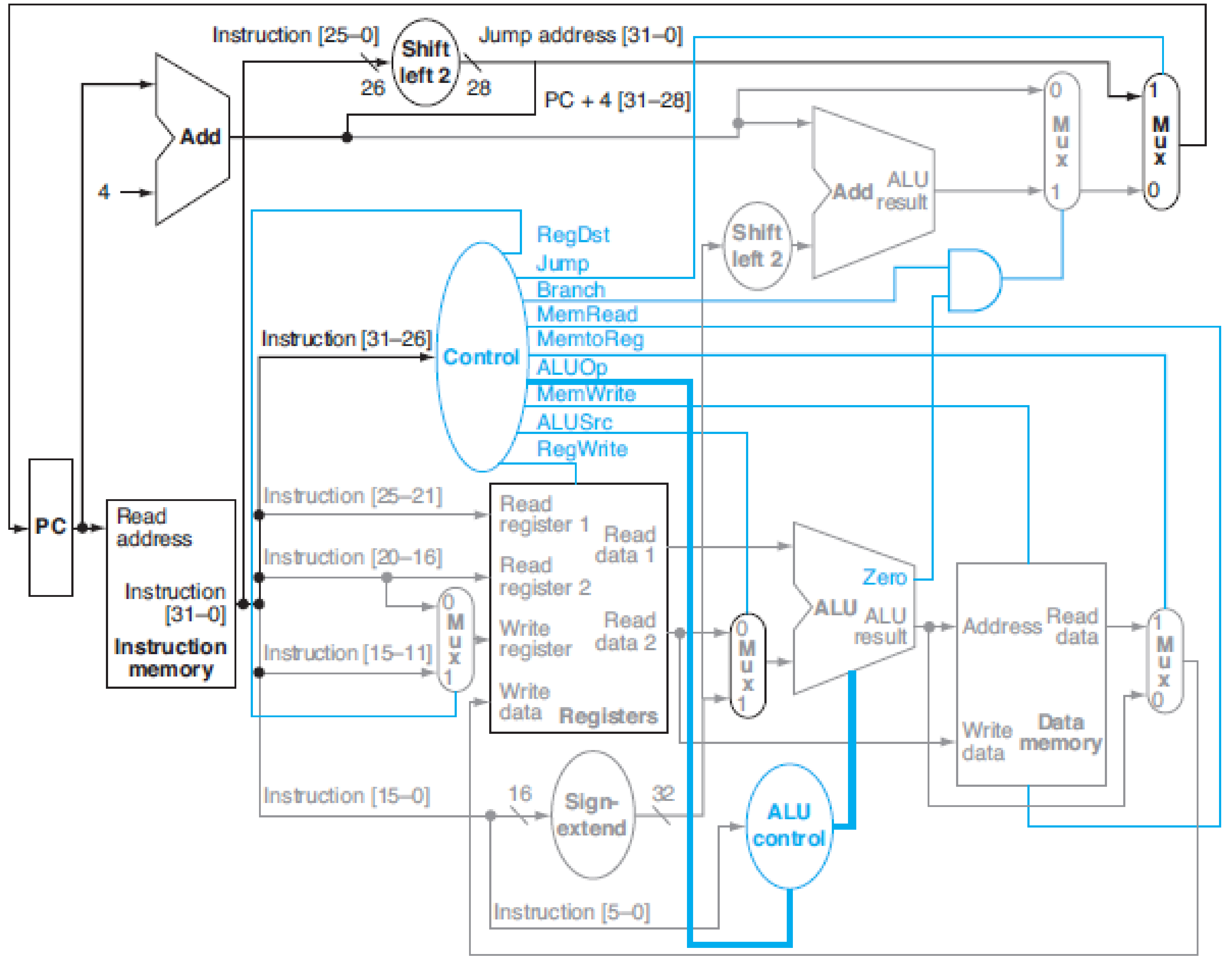

Add
1
4
Add
ALU
result
Shift
left 2
3
3
M
u
x
0
1
RegDst
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
2
2
2
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
4
PC
Read
address
Instruction
[31–0]
1
Instruction
memory
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
3
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
16
32
Sign-
extend
ALU
control
Sub

Jump



000010	address
31:26	25:0

- we can implement a jump by storing into the PC the concatenation of
  - ■ PC + 4
  - ■ the 26-bit immediate field of the jump instruction
- One additional control signal is needed for the additional multiplexor.
- This control signal, called *Jump*, is asserted only when the instruction is a jump—that is, when the opcode is 2.


Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC + 4 [31–28]
Add
4
0
M u x
1
1
M u x
0
Add
ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
0
M u x
1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
0
M u x
1
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
1
M u x
0
Instruction [15–0]
16
Sign-extend
32
ALU control
Instruction [5–0]