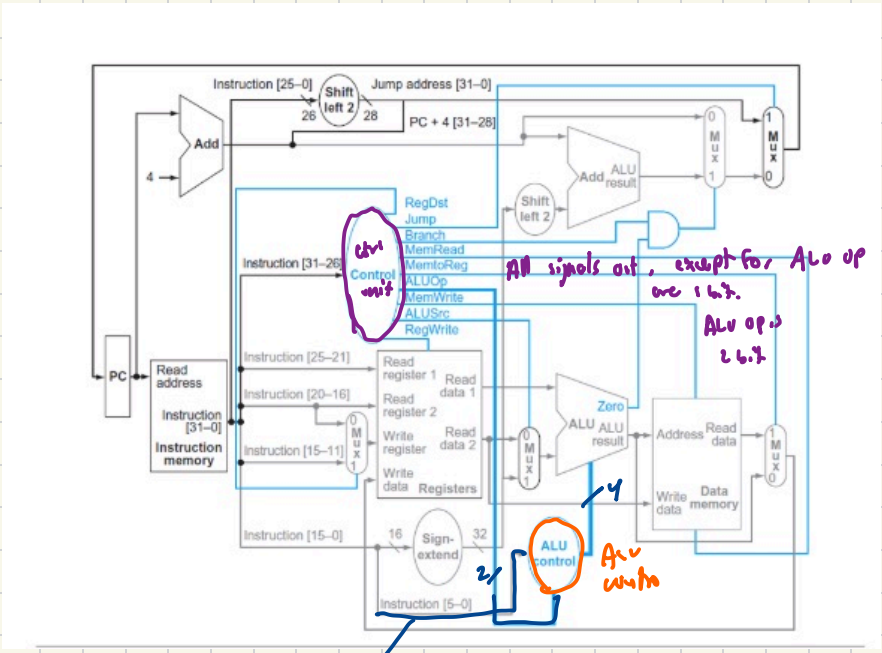


ALU control

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

Notice ALUop is never 11

R-type 5..0 has function

LOAD/STORE: Add immediate to Rs by

BRANCH EQUAL: subtract to see if equal / opposite

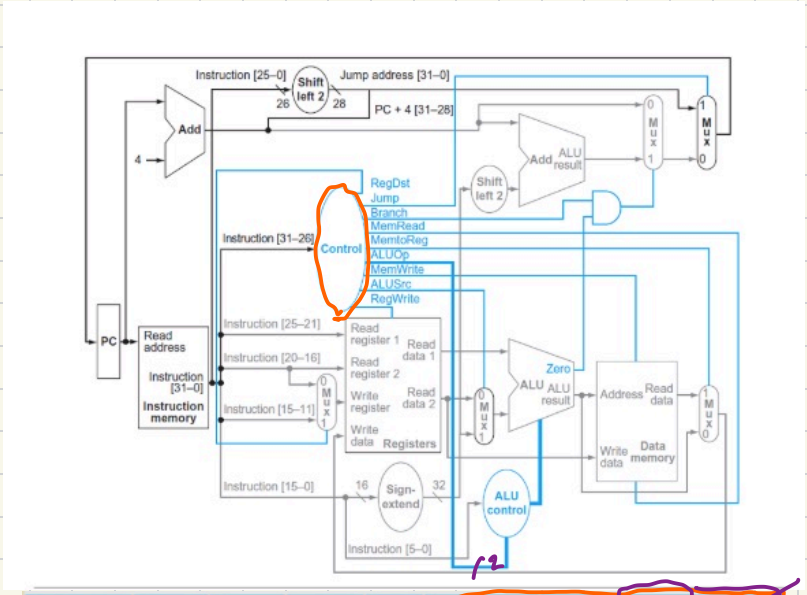
Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100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

ACTIONS

1. Add
2. subtract

Funct:

3. subtract
4. AND
5. OR
6. set on less than



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

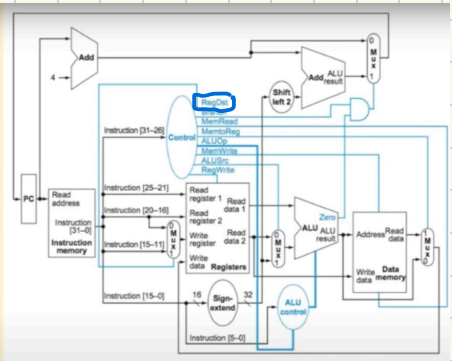
ALU op 2 bit

RegDst

controls MUX before the register file

if == 0 then register destination will be from the rt field (20:16)

if == 1 then register destination will be from the rd field (15:11)



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

4

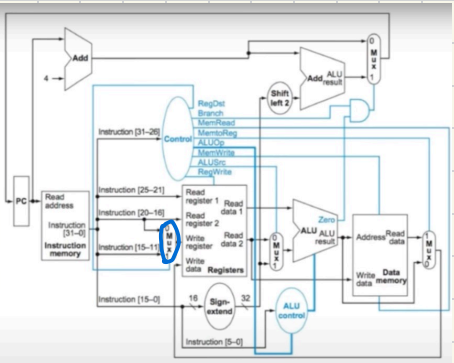
don't write back to reg file. CTRL signals don't allow, so there are just don't cares

RegWrite

control signal for writing to register

if == 0 then do not write

if == 1 then write data to register



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

5

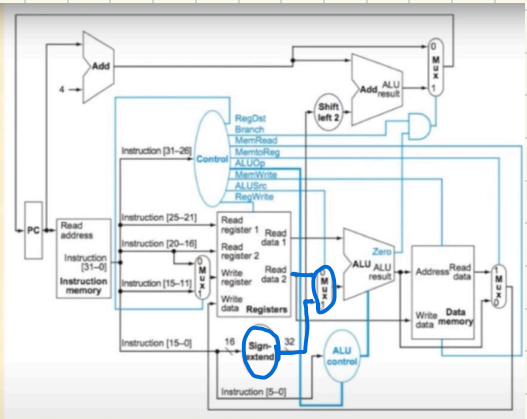
1: True
0: False

ALUSrc

controls MUX before ALU

if == 0 then second operand comes from the register file

if == 1 then second operand comes from sign-extended 16-bits of instr.



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

6

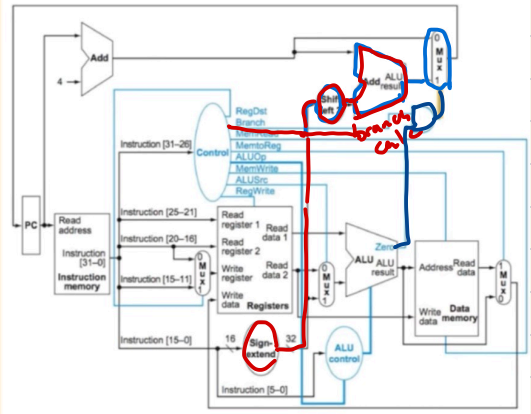
RS is first operand
Rt is second operand

PCSrc

controls MUX that updates PC

if == 0 then PC += 4

if == 1 then PC points to branch target; notice that Branch and Zero must be == 1



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

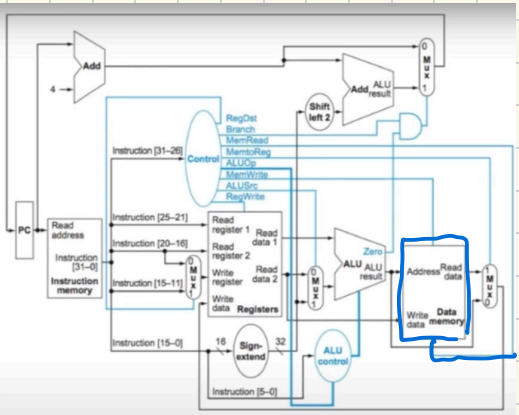
7

MemRead

control signal that allows Dmem read

if == 0 then don't read

if == 1 then read from address and place on Read data bus



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

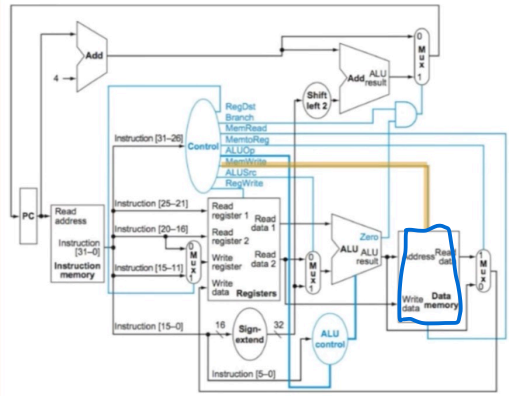
8

MemWrite

control signal for Dmem write

if == 0 then don't write

if == 1 then write data to the address from the ALU



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

9

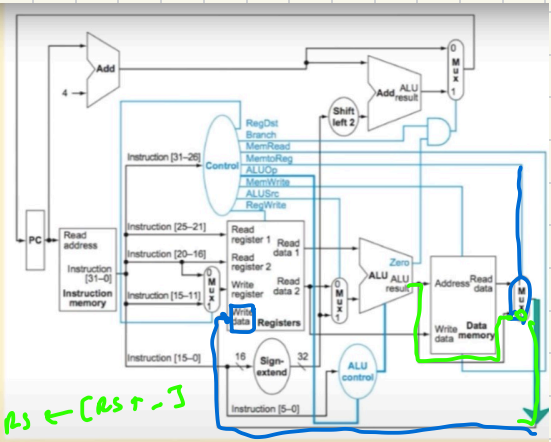
Only write to mem

MemtoReg

controls MUX at far right

if == 0 then write data from ALU to register

if == 1 then write data from Dmem to register



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

10

Truth table for main control unit

inputs are opcodes

outputs are signals

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

11

Summary

microprocessor without interleaved pipeline stages

We looked at a simple MIPS implementation that used a single long instruction cycle for every instruction.

The CPU can only do one instruction at a time.

The clock cycle has to be long enough for the slowest instruction: lw

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load word (lw)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store word (sw)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, AND, OR, slt)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

13

Time:

second :	1 s	(s)	- loading an app, compiling
millisecond :	$1 \cdot 10^{-3}$ s	(ms)	- disk seek, keyboard input
microsecond :	$1 \cdot 10^{-6}$ s	(µs)	- interrupts, system calls
nanosecond :	$1 \cdot 10^{-9}$ s	(ns)	- Ram access, cpu cycles
picosecond :	$1 \cdot 10^{-12}$ s	(ps)	- gate delays, signal timing

Don't memorize, think!

- how it performs operation
- does it access memory
- does it branch
- does it write to mem

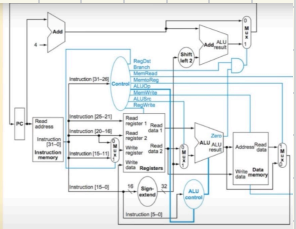
Main control output

MemWrite RegDst

MemRead ALUSrc

RegWrite MemtoReg

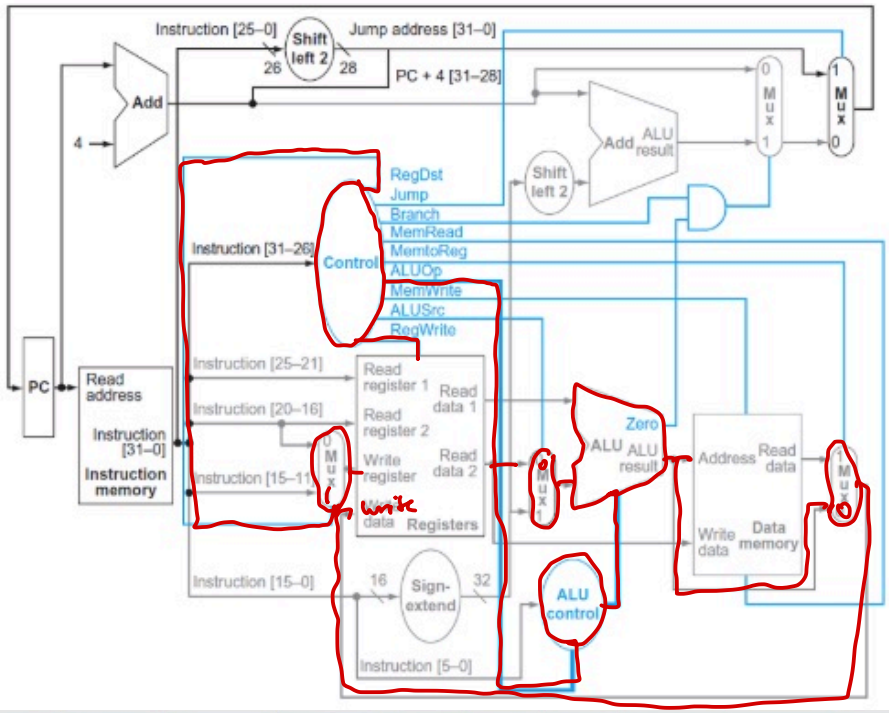
Branch



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

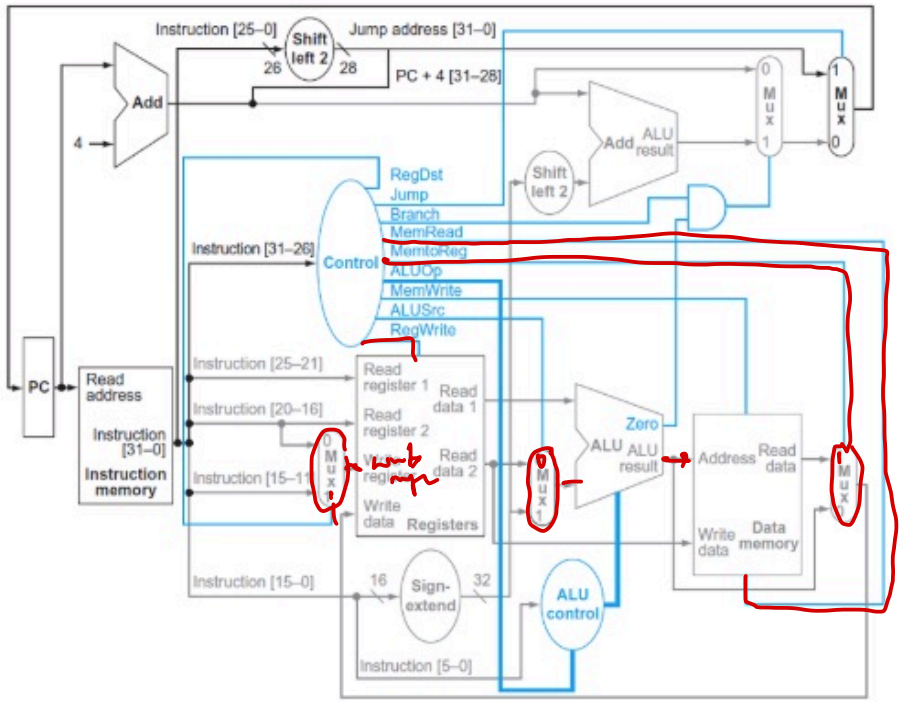
14

R-type Instruction



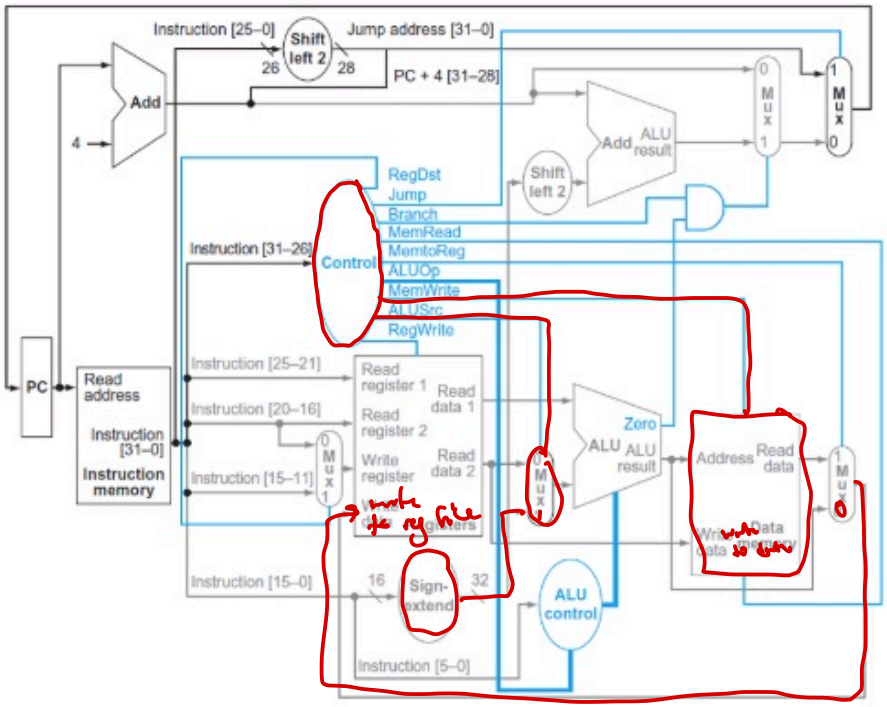
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Load Instruction



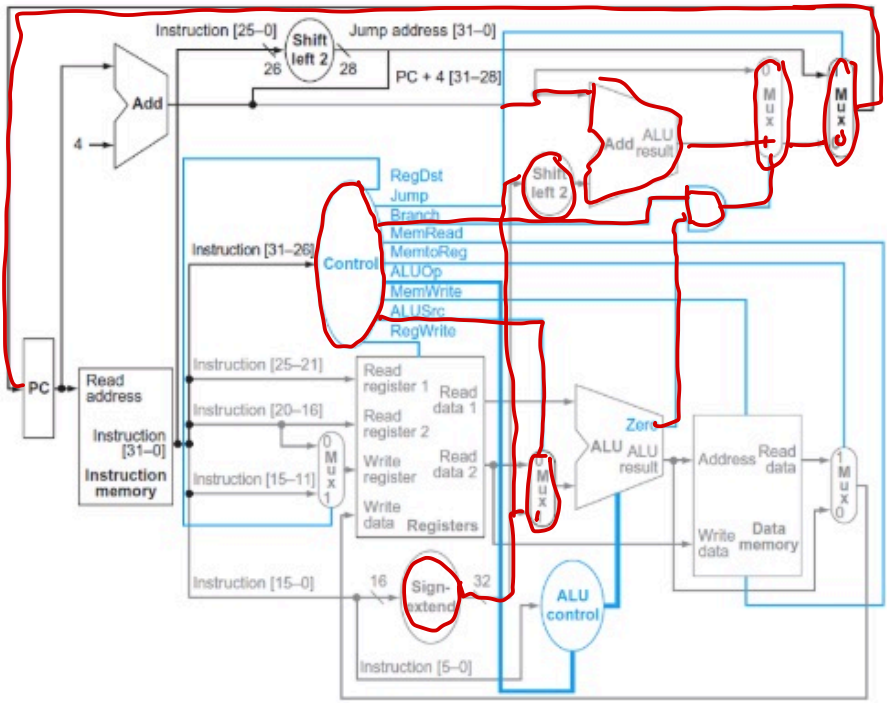
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Store Instruction

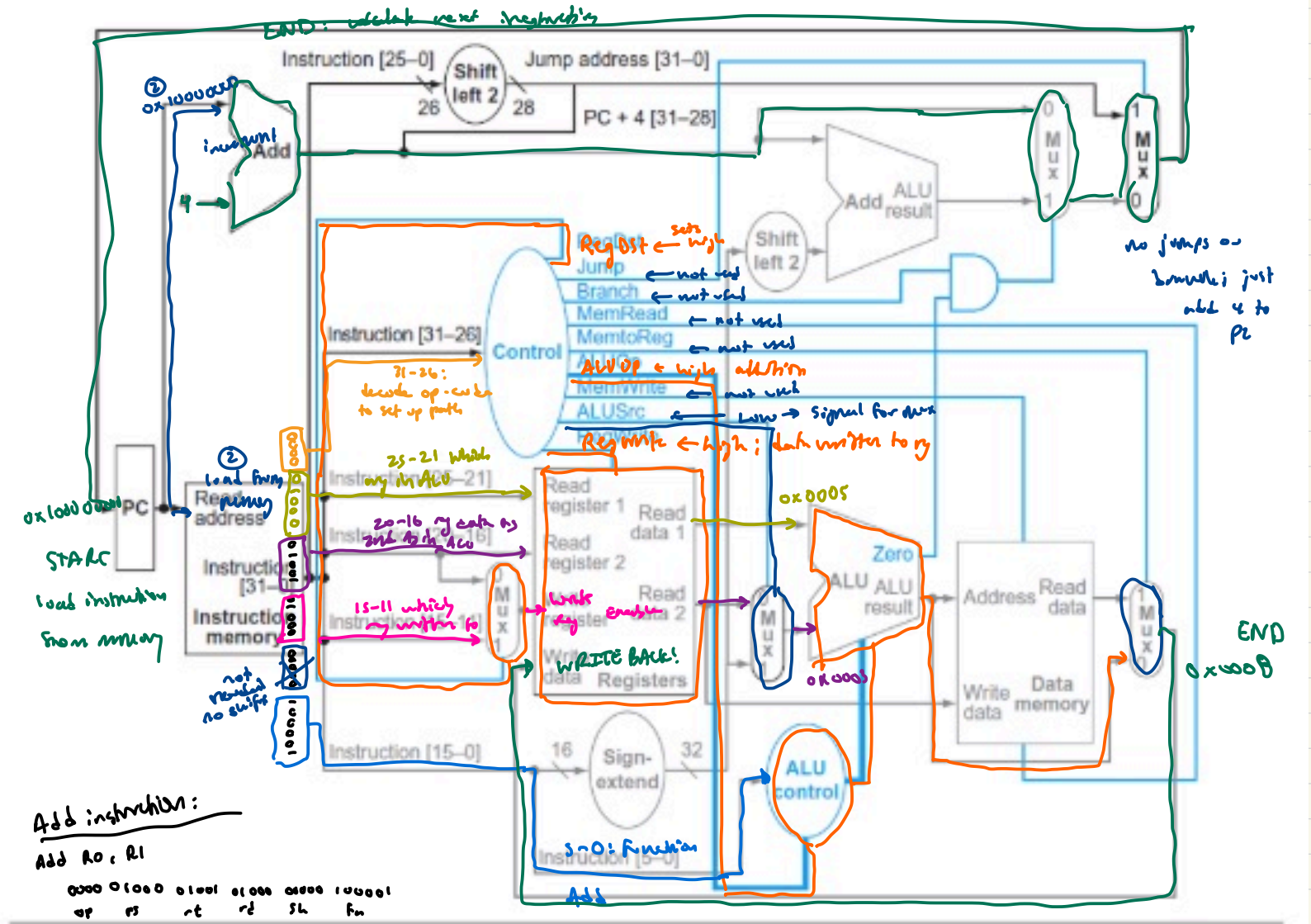


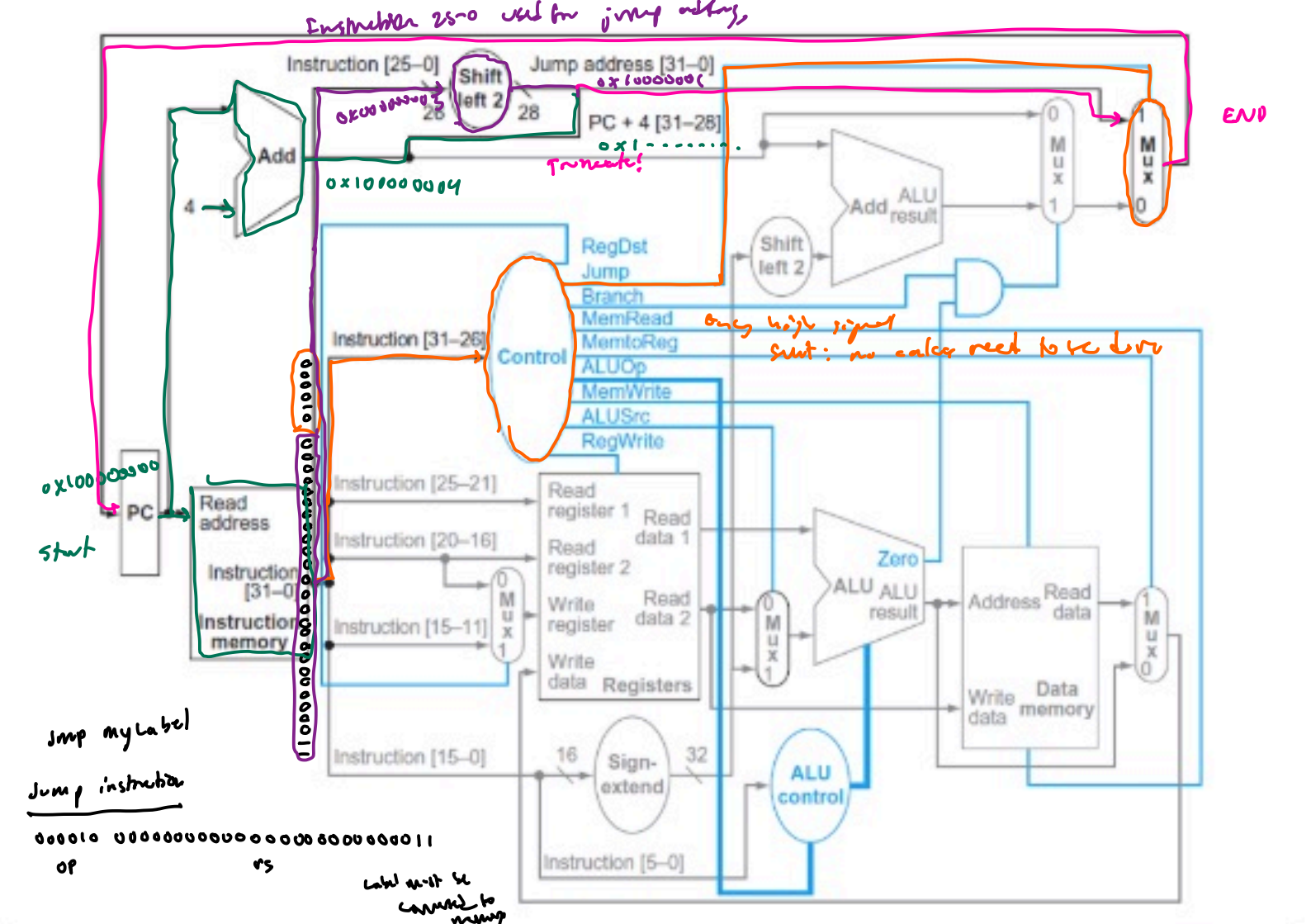
Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

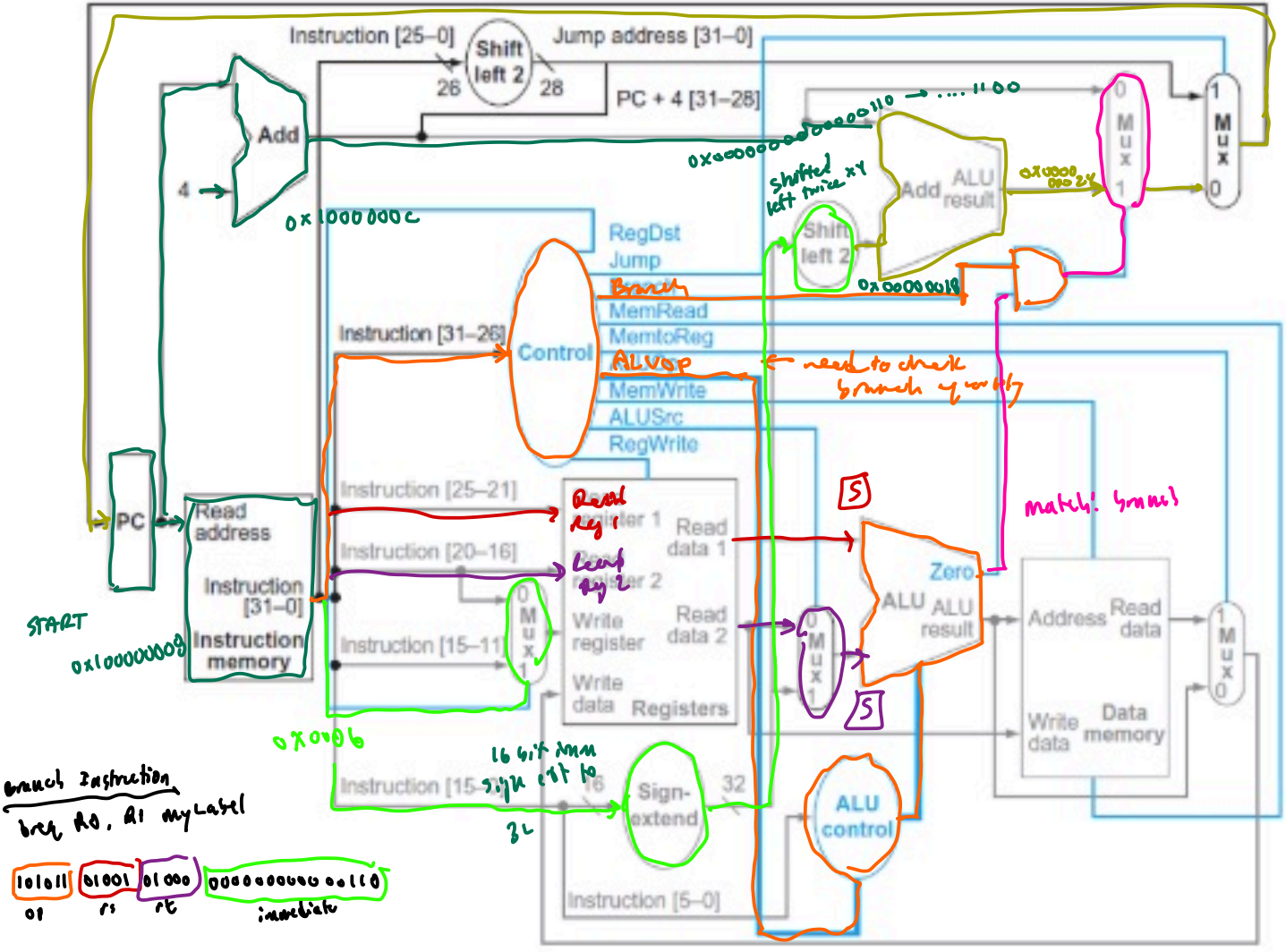
Branch Instruction

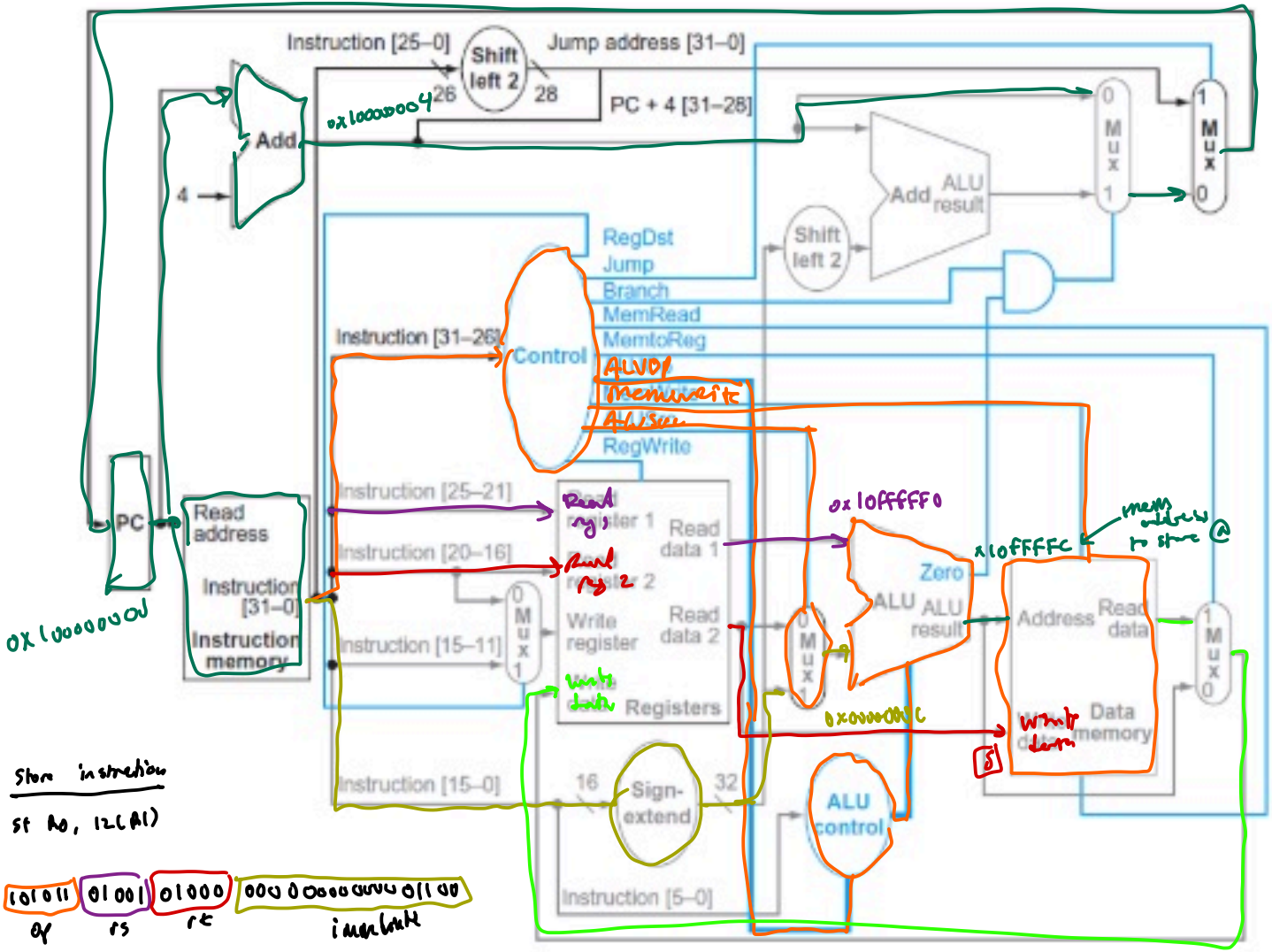


Instruction	RegDst	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

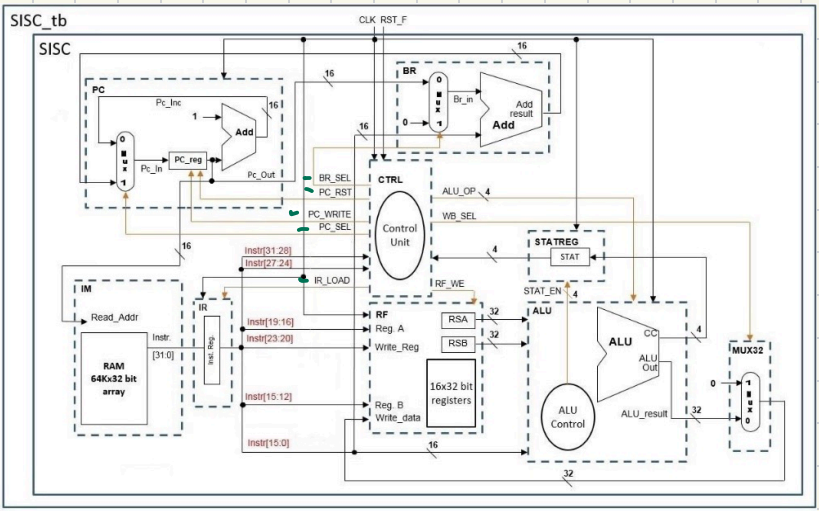

END: calculate next instruction
Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
Add
ALU result
M u x
RegDst ← sets high
Jump ← not used
Branch ← not used
MemRead ← not used
MemtoReg ← not used
ALUOp ← tells ALU this
MemWrite ← not used
ALUSrc ← Low → signal for data
RegWrite ← high; data written to rg
Instruction [31–26]
Control
31–26: decode op-code to set up paths
25–21 which
20–16 rg data as 2nd to ALU
15–11 which rg written to
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
WRITE BACK!
0x0005
Zero
ALU
ALU result
0x0001
Address
Read data
Write data
Data memory
Sign-extend
16
32
ALU control
PC
Instruction memory
Add
4
not needed no shifts
no jumps or branch; just add 4 to PC
END
0x0008
START
load instruction from memory
Add instruction:
Add R0, R1
0000 01000 01001 01000 00000 100001
op rs rt rd sh fn
Add


Instruction [25–0] used for jump address
Instruction [25–0]
Shift left 2
Jump address [31–0]
0x1000000(
0x10000004
28
28
PC + 4 [31–28]
0x1.....
Truncate!
END
Add
4
0x10000004
Add
ALU result
Shift left 2
M u x
0
1
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
only keep signal
hint: no value need to be store
0x10000000
PC
start
Read address
Instruction [31–0]
Instruction memory
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
Zero
ALU
ALU result
Address
Read data
Write data
Data memory
16
Sign-extend
32
ALU control
jump myLabel
Jump instruction
000010 00000000000000000000000011
op
rs
label must be converted to memory


Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC + 4 [31–28]
Add
4
0x10000000
0x00000002
Shifted left twice
Add
ALU result
0x00000018
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
need to check branch or not
PC
Read address
Instruction [31–0]
Instruction memory
START
0x10000000
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
Zero
ALU
ALU result
match! branch
Address
Read data
Write data
Data memory
Sign-extend
16
32
16 bit imm
sign ext to 32
ALU control
0x0006
Branch Instruction
beq R0, R1 myLabel
000100
01001
01000
0000000000000110
op
rs
rt
immediate


Instruction [25–0]
Shift left 2
Jump address [31–0]
26
28
PC + 4 [31–28]
0x10000004
Add
4
M u x
0
1
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]
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
ALU control
Zero
ALU
ALU result
0x10fffff0
0x10fffffc
mem address to store @
0x00000004
Address
Read data
Write data
Data memory
0x(00000000)
Store instruction
sf R0, 12(R1)
101011
01001
01000
0000 0000 0000 1100
op
rs
rt
immediate

Back to comp Arch Diagram



Given:

1. instructions contained inside imem.data
2. verilog files pc.v, br.v, ir.v, & im.v DO not modify
3. testbench file that only generates clock and reset signals

Todo:

1. modify control unit to generate BR_SEL, PC_RST, PC_WRITE, PC_SEL, IR_LOAD
2. modify SISC module to instantiate and connect the pc, br, ir, im modules
3. compress project folder and submit

Imem.data:

- upon starting simulation, im module reads data inside file: imem.data
- data is written as 32-bit hexidecimal strings (0x0000). Data sequential from address 0x0000...
- this handles timing of instructions with delays. Remove elsewhere.
- These instructions are read during instruction fetch stage of pipeline

1. Fetch
2. Decode
3. Compute
4. Memory
5. Write

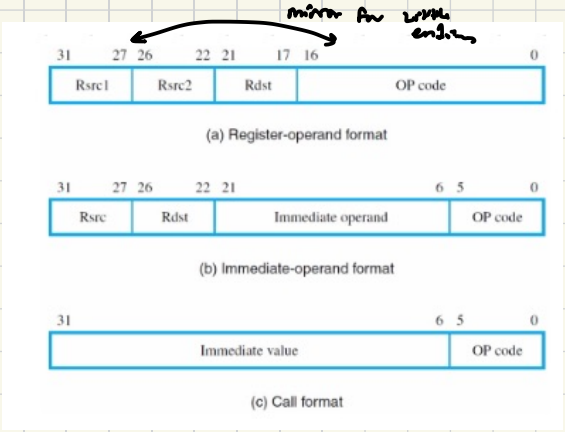
PC update:

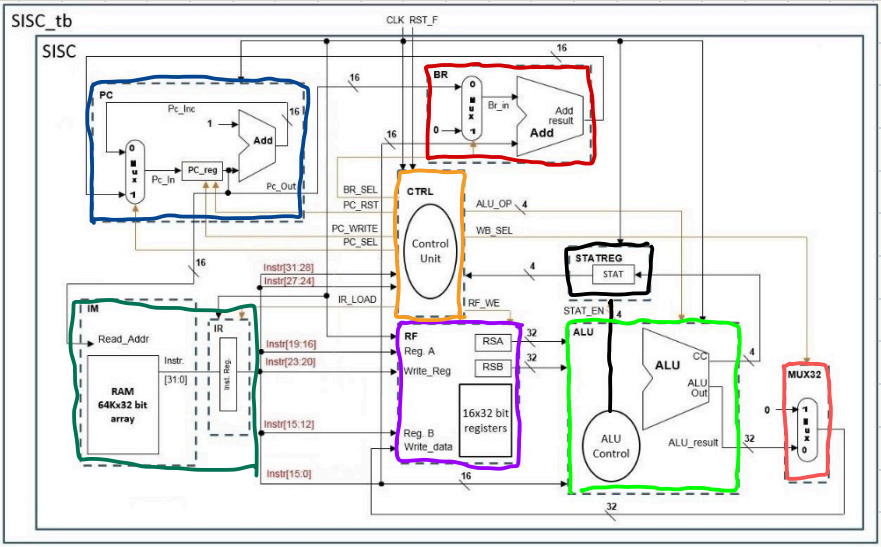
- PC value incremented as part of instruction fetch stage.
- output is fed into BR module for use in decode stage to determine if branch is taken
- Implement BR inside of decode stage

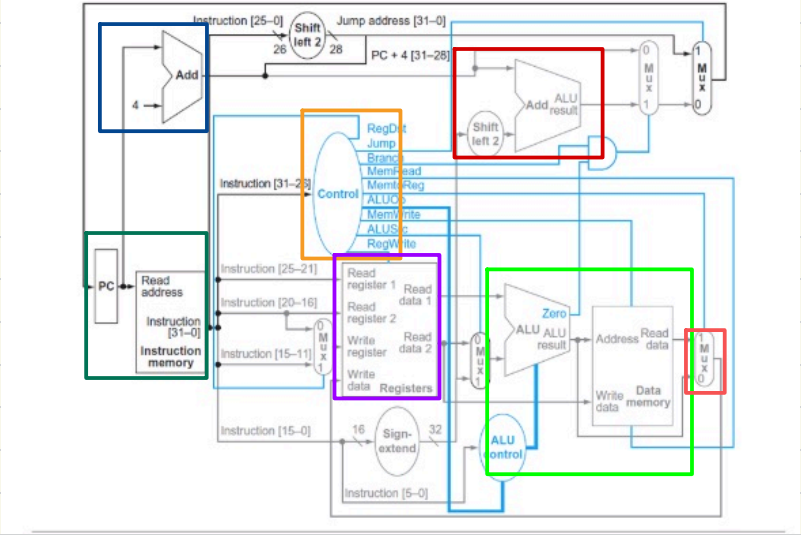
Little endian : LSB @ lowest

operation	opcode	Format
nop	0x00	-
ADI	0x21	immediate operand (b)
ADD	0x11	register operand (a)
SHL	0x1B	register operand (a)
SUB	0x12	register operand (a)
SHR	0x1A	register operand (a)
XOR	0x17	register operand (a)
NOT	0x14	register operand (a)
ROL	0x19	register operand (a)
OR	0x15	register operand (a)
AND	0x16	register operand (a)
ROR	0x18	register operand (a)
BNE	0x61	register operand (a)
BRR	0x51	immediate operand (b)
BNR	0x71	register operand (a)
HALT	0xF0	-

mirror for little endian







many small differences...