ELE 455/555 Computer System Engineering

Section 2 – The Processor

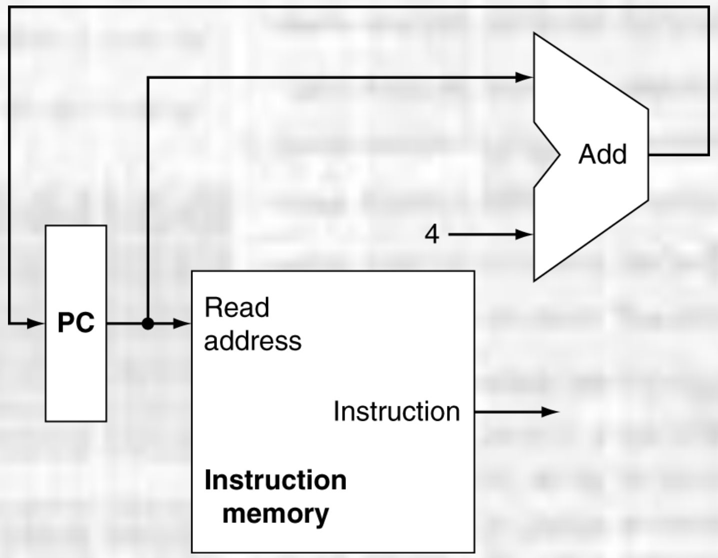
Class 2 – Simple Data Path

Simple Data Path
Overview

- 5 Stages of Instruction Execution
  - Fetch (IF)
  - Decode / Register Access (ID)
  - Execute (EX)
  - Memory Access (MEM)
  - Write Back (WB)
- Everything is asynchronous except for the PC
  - PC is a synchronous register
    - Positive edge triggered
    - Synchronous reset

Simple Data Path
Overview

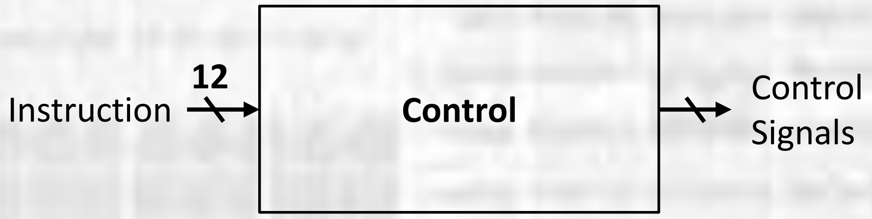
- Instruction Fetch
  - Clock the PC
    - New address is provided to the memory
    - Memory provides instruction to its output
    - Next address is provided to PC input
      - Memory is Byte Addressed
      - Instructions are 4 bytes wide
      - → increment by 4
    - Adder is drawn as an ALU but actual implementation would be our optimized adder block

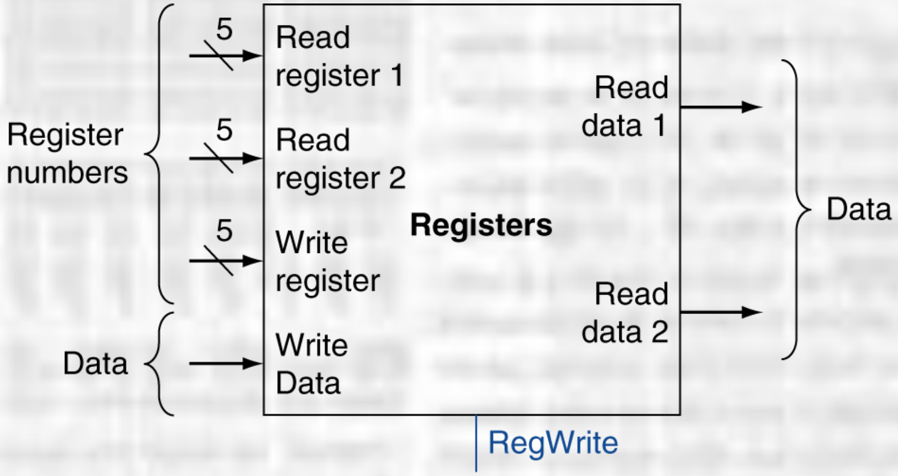


Simple Data Path

Overview

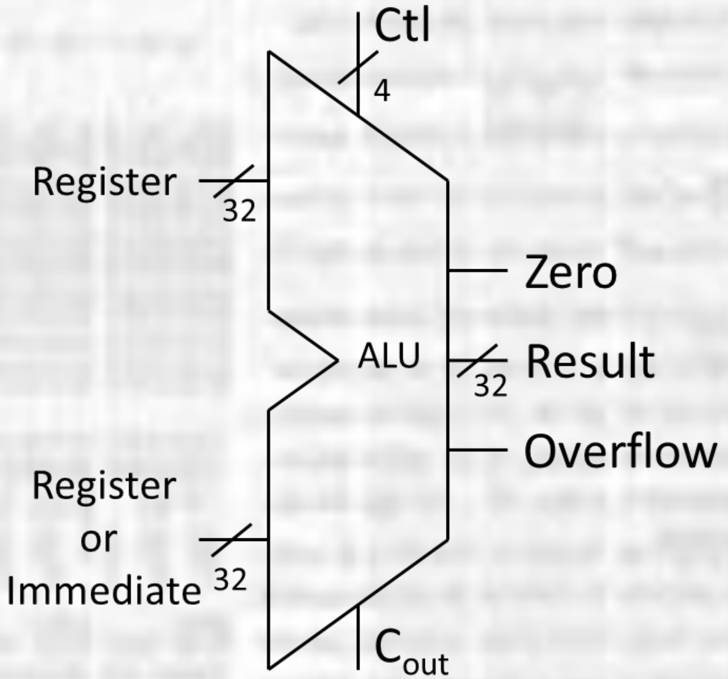
- Decode / Register Access
  - Decode
    - For MIPS uses first and last 6 bits of the instruction
  - Register Access
    - R format instructions use at most:
      2 source registers and
      1 destination register
    - I format instructions use:
      immediate: 1 src, 1 dest
      load/store: 1 src or 1 dest
      branch: 2 src
    - J format instructions do not use registers





Simple Data Path
Overview

- Execute
  - ALU executes all arithmetic and logical instructions
  - Inputs are Registers or Immediates
    - Immediates are contained in the instruction



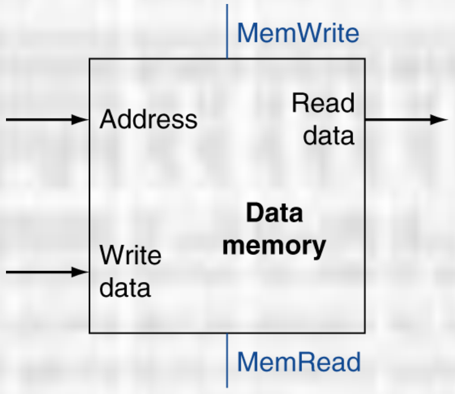
Simple Data Path

Overview

- Memory Access

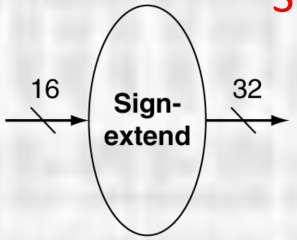
  - Load / Store Instructions
    lw $t4,4($t0) # load $t4 from memory location ($t0)+4

  - Address is calculated by adding the offset to the value in a register
    - Use the ALU to add register value to the offset
    - Since the offset is only 16 bits and is in 2's compliment format
      - Must sign extend the offset to 32 bits



a. Data memory unit

Sign extension is trivial – Why?

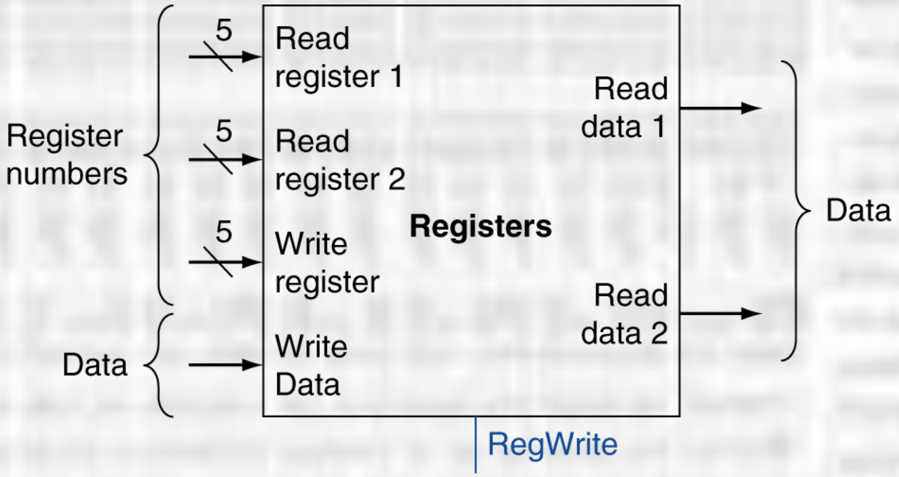


b. Sign extension unit

Simple Data Path

Overview

- Write Back
  - Write results or memory value back to a register
  - Write data comes from ALU (result)
    or
  - Write data comes from data memory



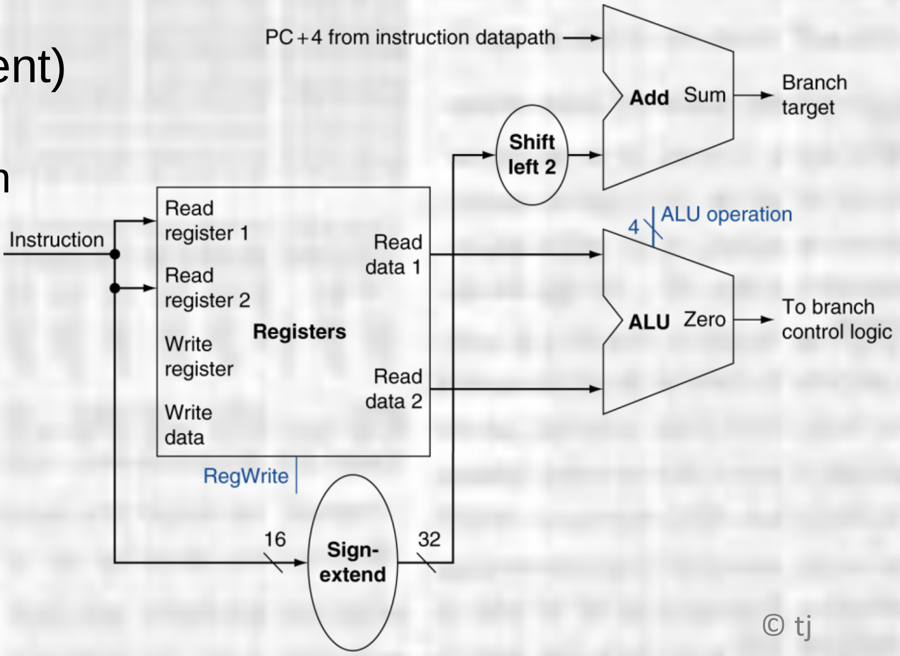
Simple Data Path

Overview

- Missing Pieces – branches
  - Read register operands
  - Compare operands
    - Use ALU, subtract and check Zero output
  - Calculate target address
    - Sign-extend displacement
    - Shift left 2 places (word displacement)
    - Add to PC + 4
      - Already calculated by instruction fetch

shift left 2 is trivial – why?

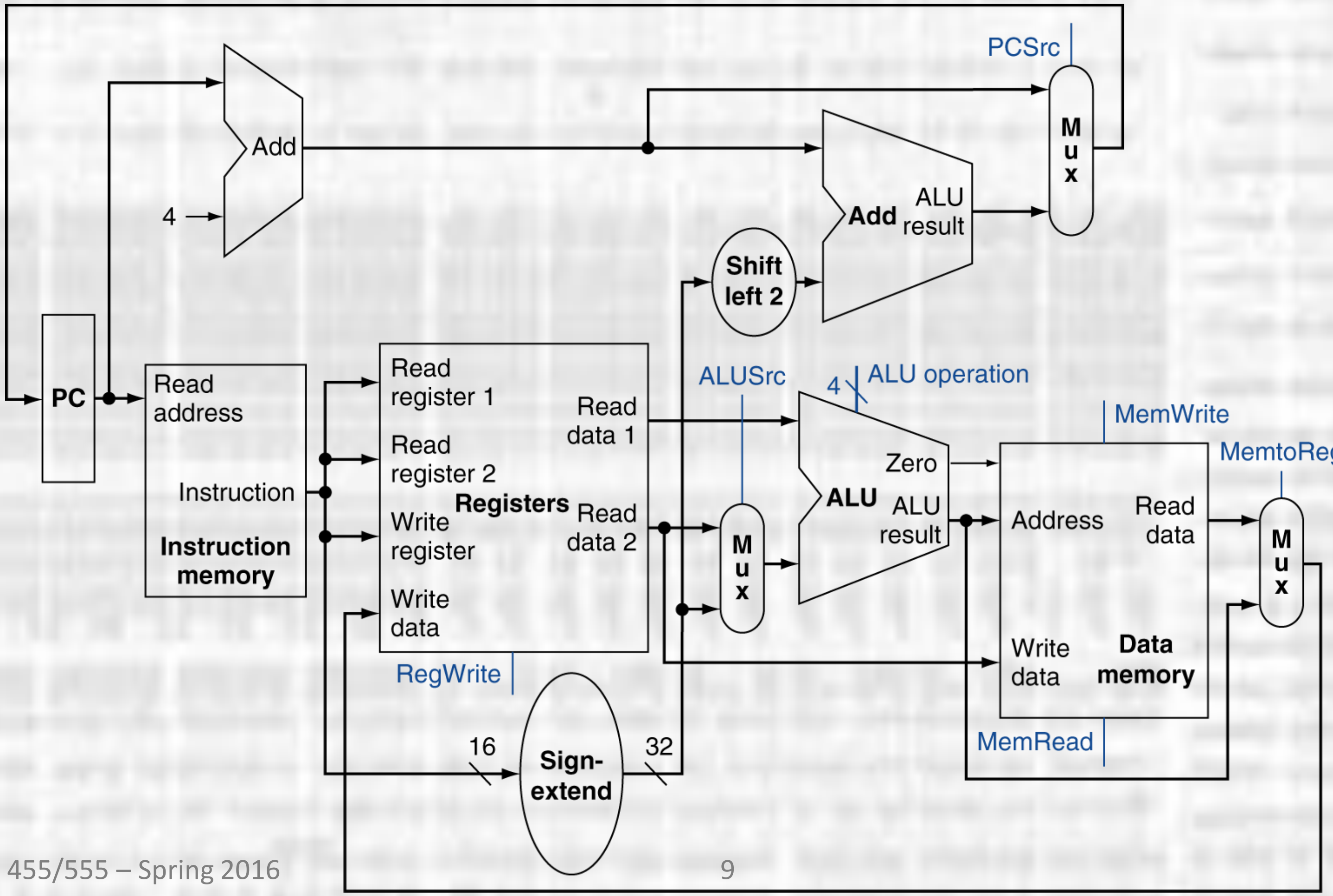
What about the bits that shift off the end?



Simple Data Path

Overview

- Full Datapath



Simple Data Path

Control

- ALU Control

  - Basic ALU control mapping
  - Slightly different MUX wiring than our previous version

Operation	invA	negB	ctl[1]	ctl[0]
AND	0	0	0	0
OR	0	0	0	1
NOR	1	1	0	0
ADD	0	0	1	0
SUB	0	1	1	0
SLT	0	1	1	1

Simple Data Path

Control

- ALU Control

  - LW and SW use the ALU to add an offset to a register value
  - BEQ uses the ALU to do a subtract
  - R-type instructions can do any of the ALU functions

  - Create an ALU opcode to generate the ALU control signals based on the instruction being executed

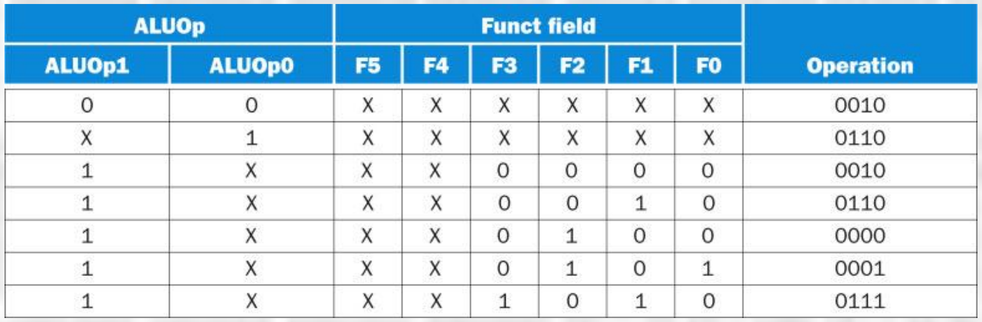
opcode	ALUOp	Operation	funct	ALU function	ALU control
lw	00	load word	XXXXXX	add	0010
sw	00	store word	XXXXXX	add	0010
beq	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001
		set-on-less-than	101010	set-on-less-than	0111

Simple Data Path

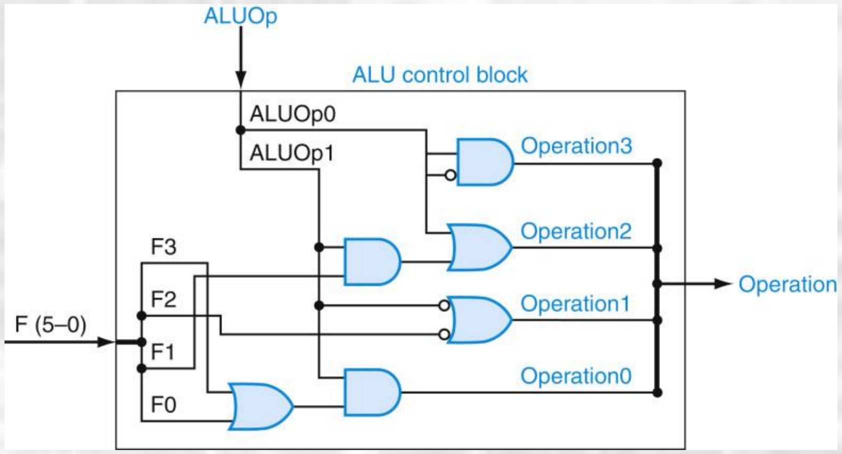
Control

- ALU Control

  - Creating the logic starts with a truth table
  - Note there are many “don’t care” states



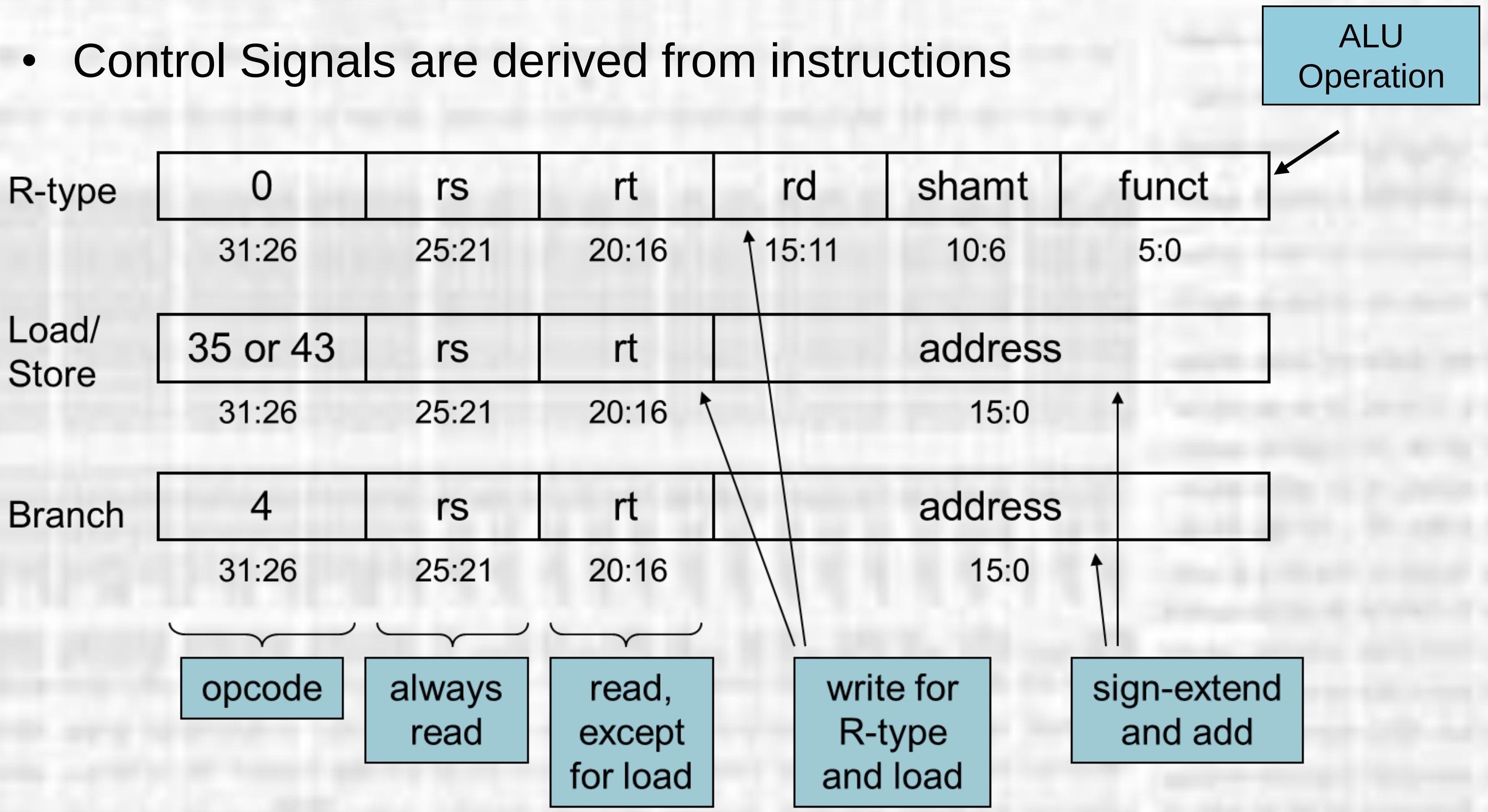
ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111



Simple Data Path

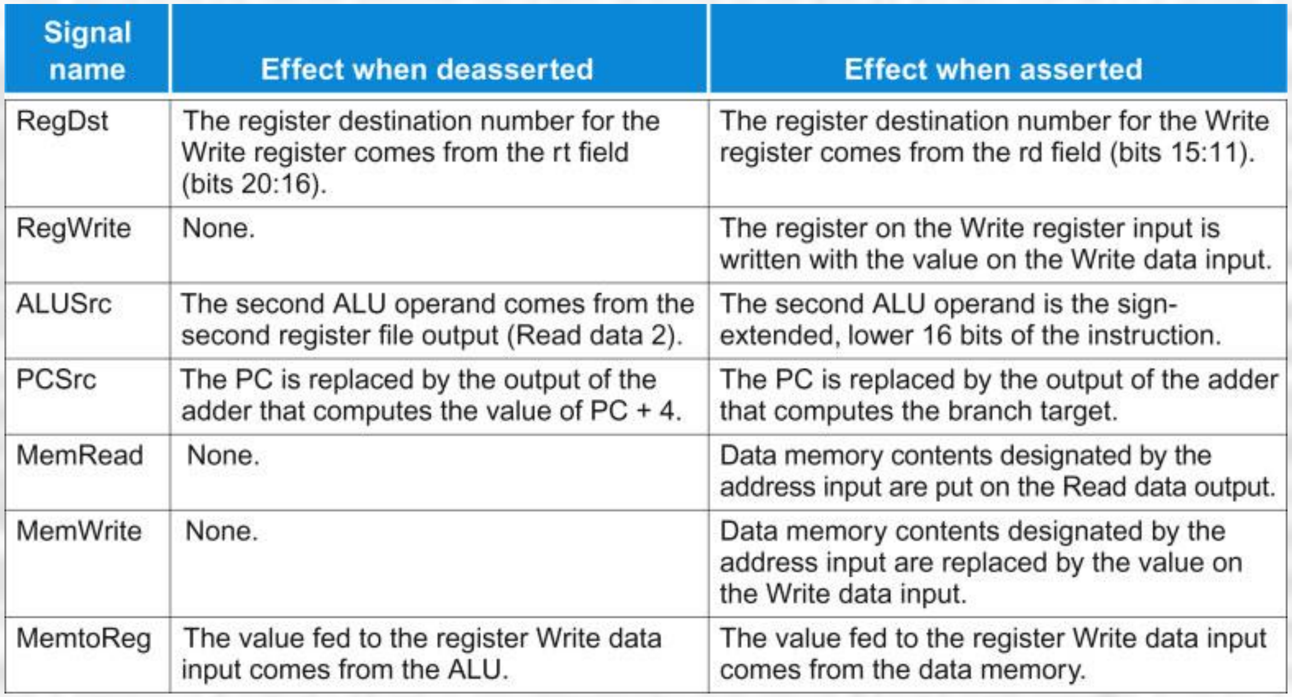
Control

- Datapath Control
  - Control Signals are derived from instructions



Simple Data Path
Control

- Datapath Control
  - Control Signals are derived from instructions

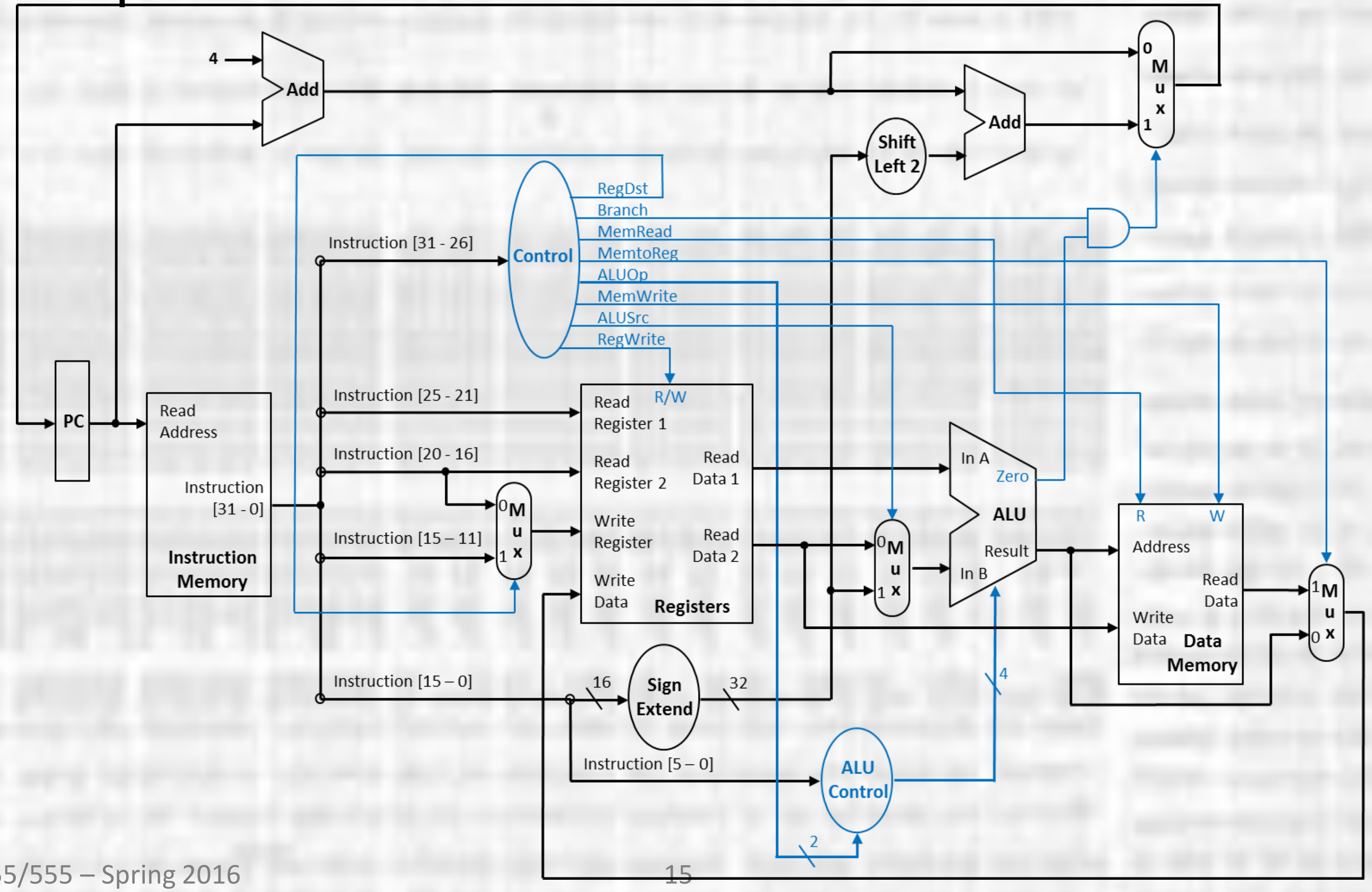


Signal name	Effect when deasserted	Effect when asserted
RegDst	The register destination number for the Write register comes from the rt field (bits 20:16).	The register destination number for the Write register comes from the rd field (bits 15:11).
RegWrite	None.	The register on the Write register input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

Simple Data Path

Control

- Datapath Control



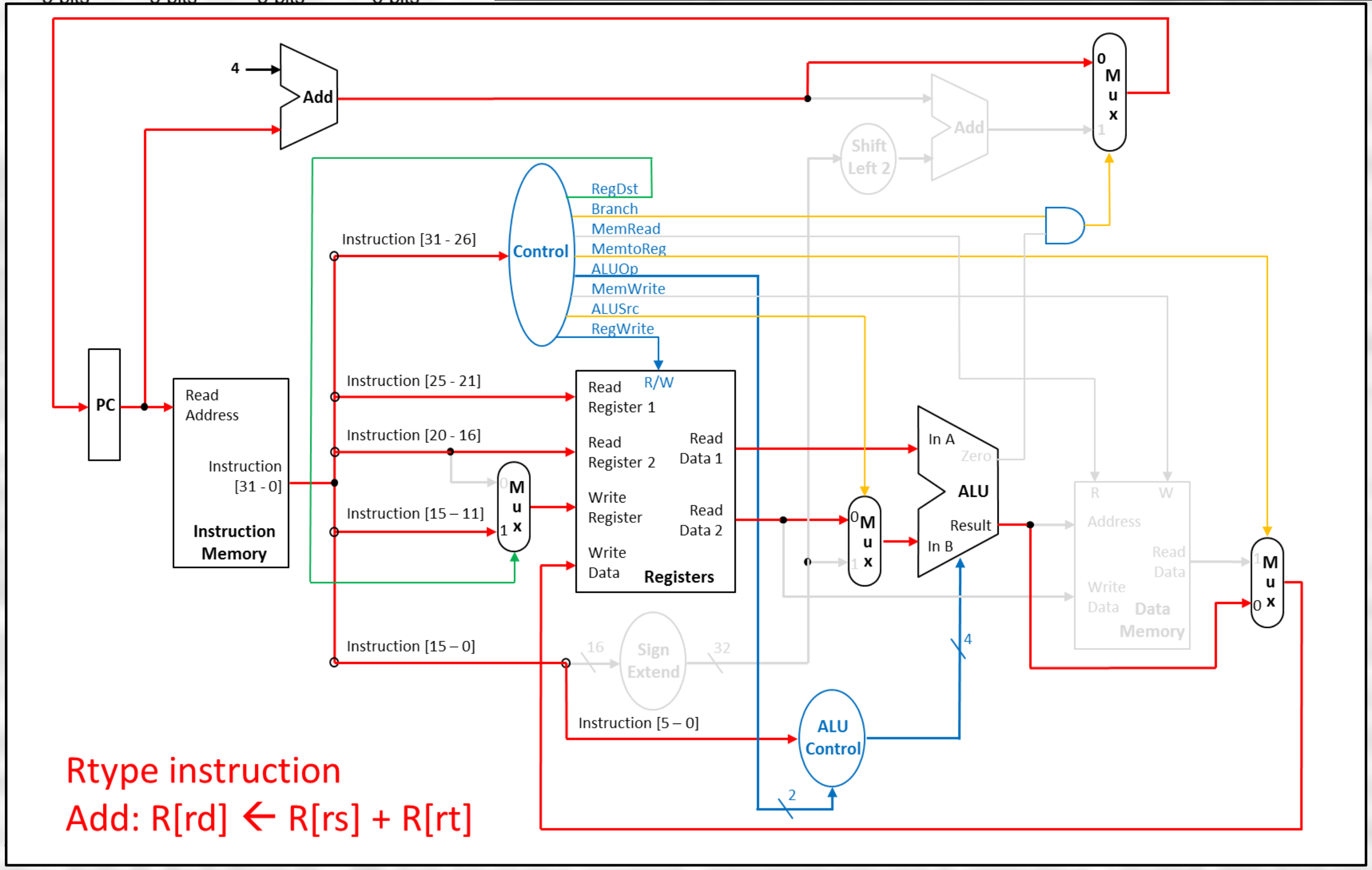
Simple Data Path

Control

- Datapath Control – Rtype Instruction

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

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0



Rtype instruction
Add: R[rd] ← R[rs] + R[rt]

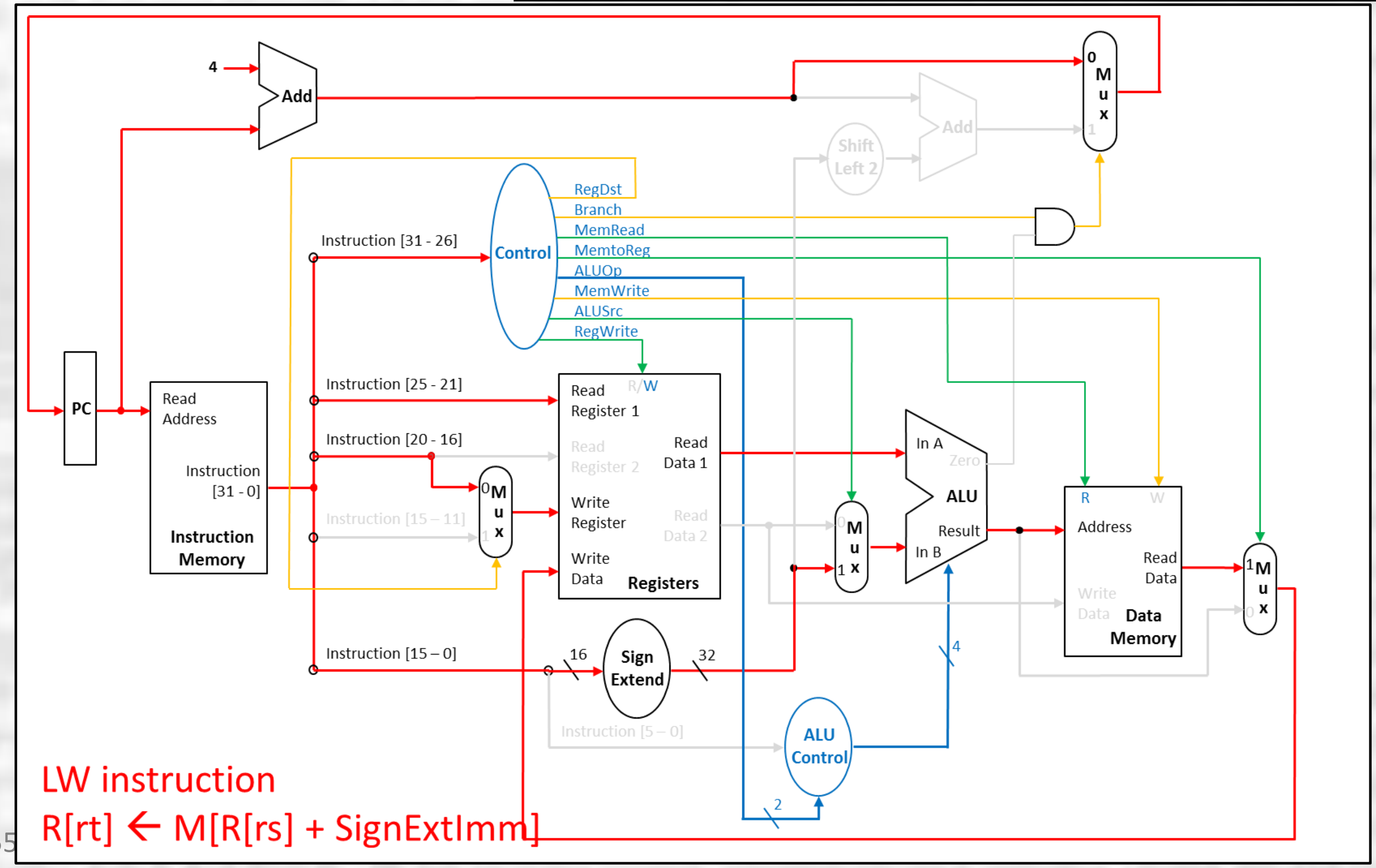
Simple Data Path

Control

- Datapath Control – LW Instruction

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

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
LW	0	1	1	1	1	0	0	0	0



LW instruction
R[rt] ← M[R[rs] + SignExtImm]

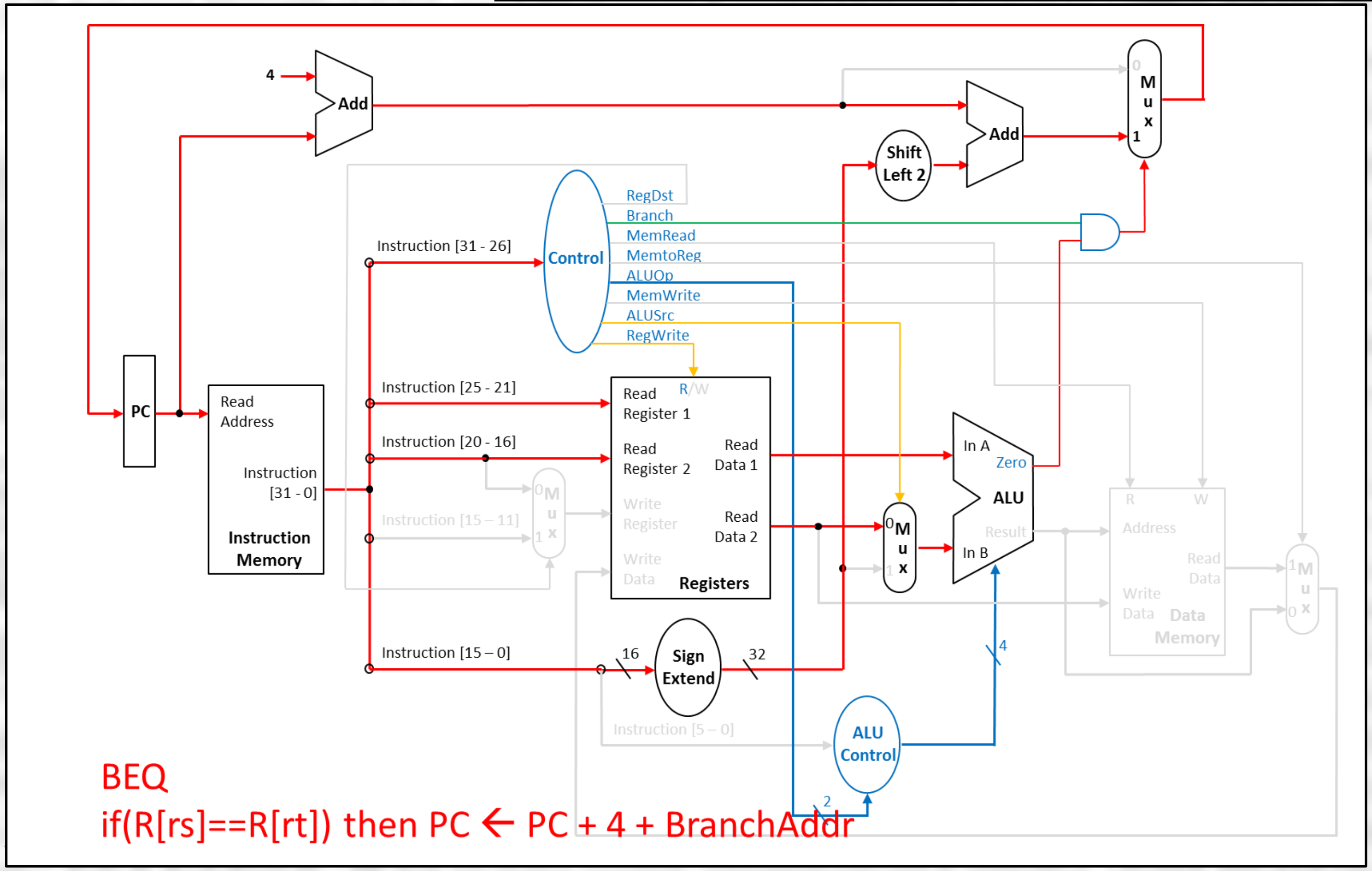
Simple Data Path

Control

- Datapath Control – BEQ

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

Instruction	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
LW	X	0	X	0	0	0	1	0	1



BEQ
if(R[rs]==R[rt]) then PC ← PC + 4 + BranchAddr

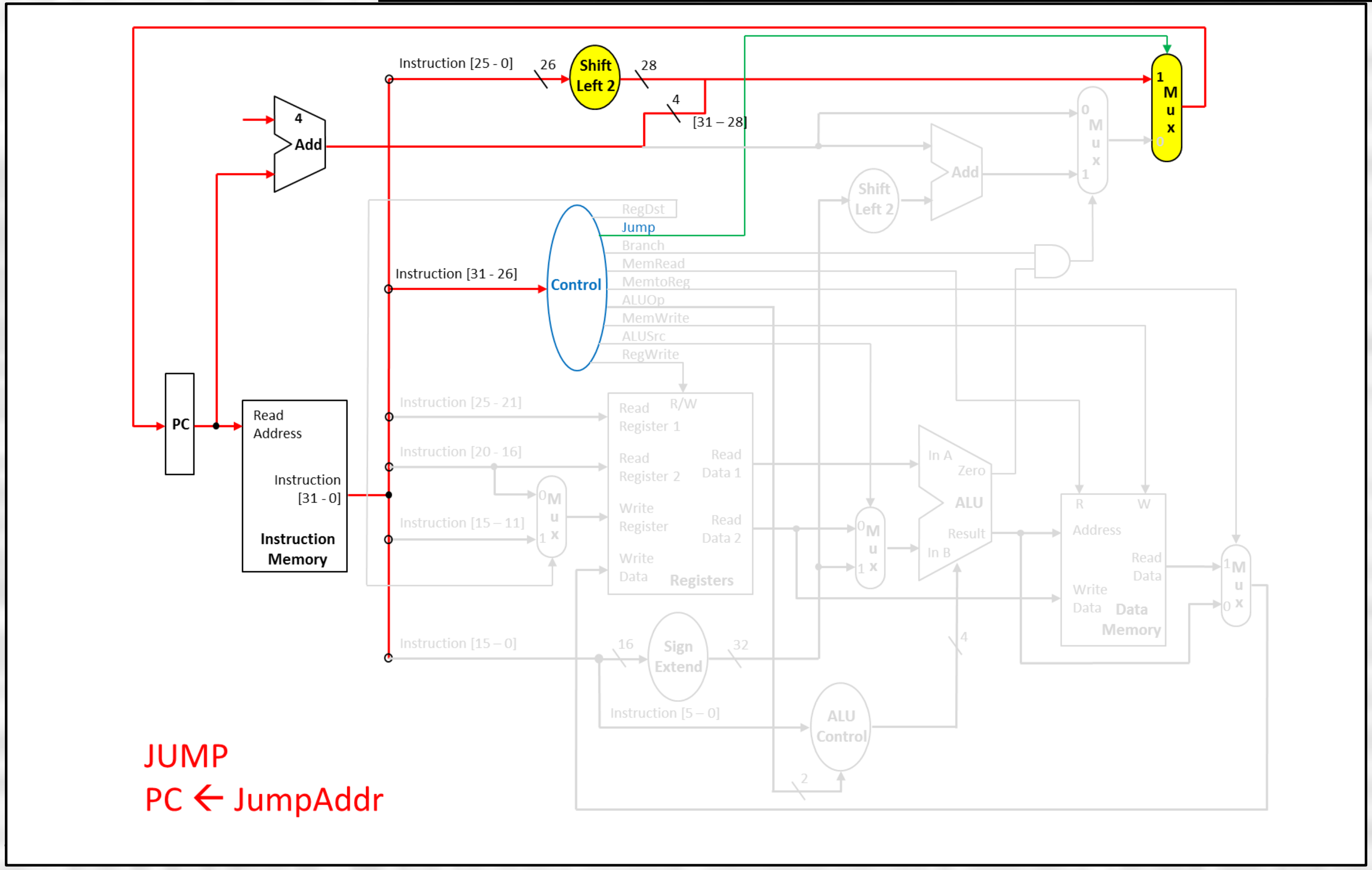
Simple Data Path

Control

- Datapath Control – JUMP

op	address
6 bits	26 bits

Instruction	Jump	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
J	1	X	X	X	0	X	0	X	X	X



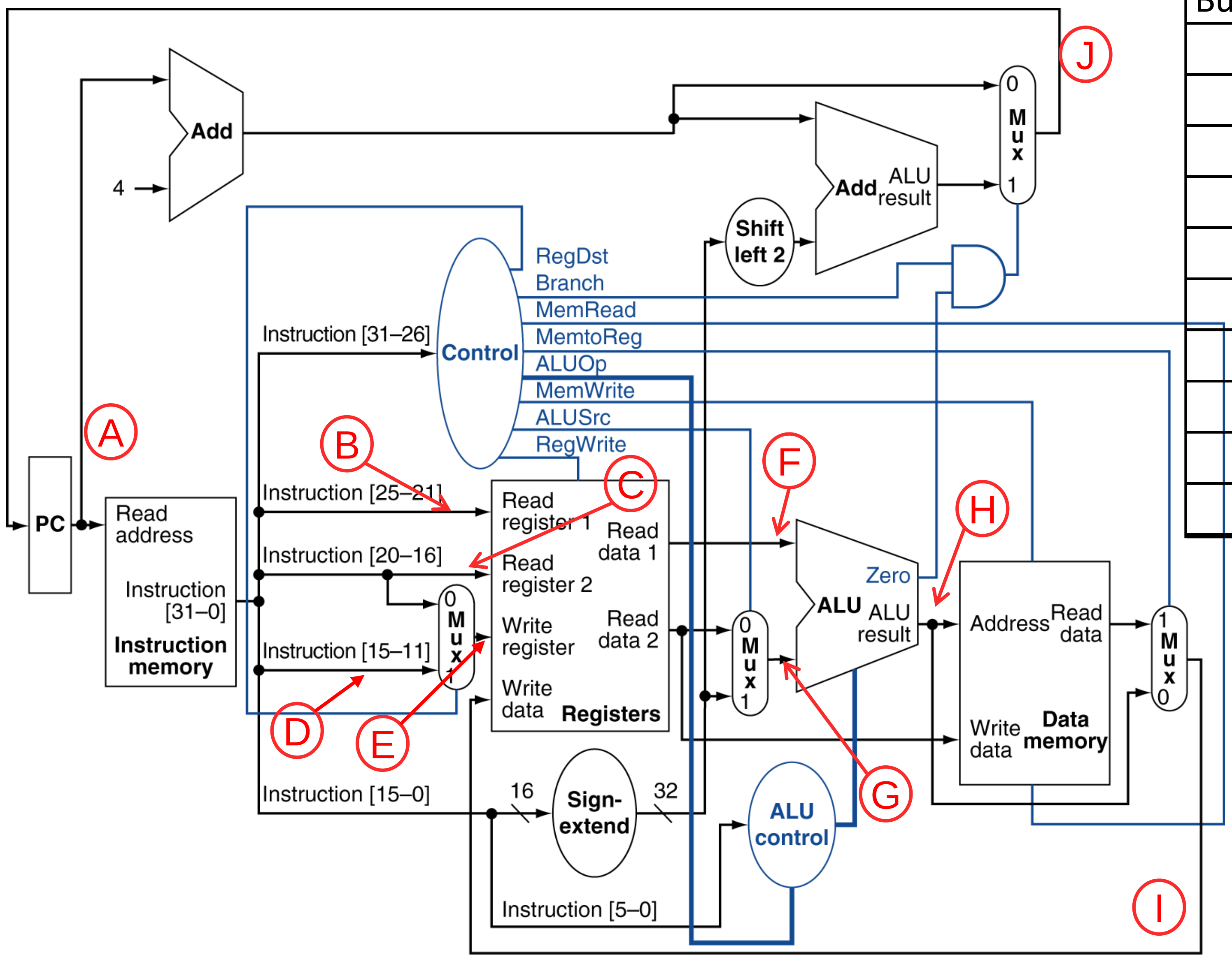
JUMP
PC ← JumpAddr

Simple Data Path

Issues

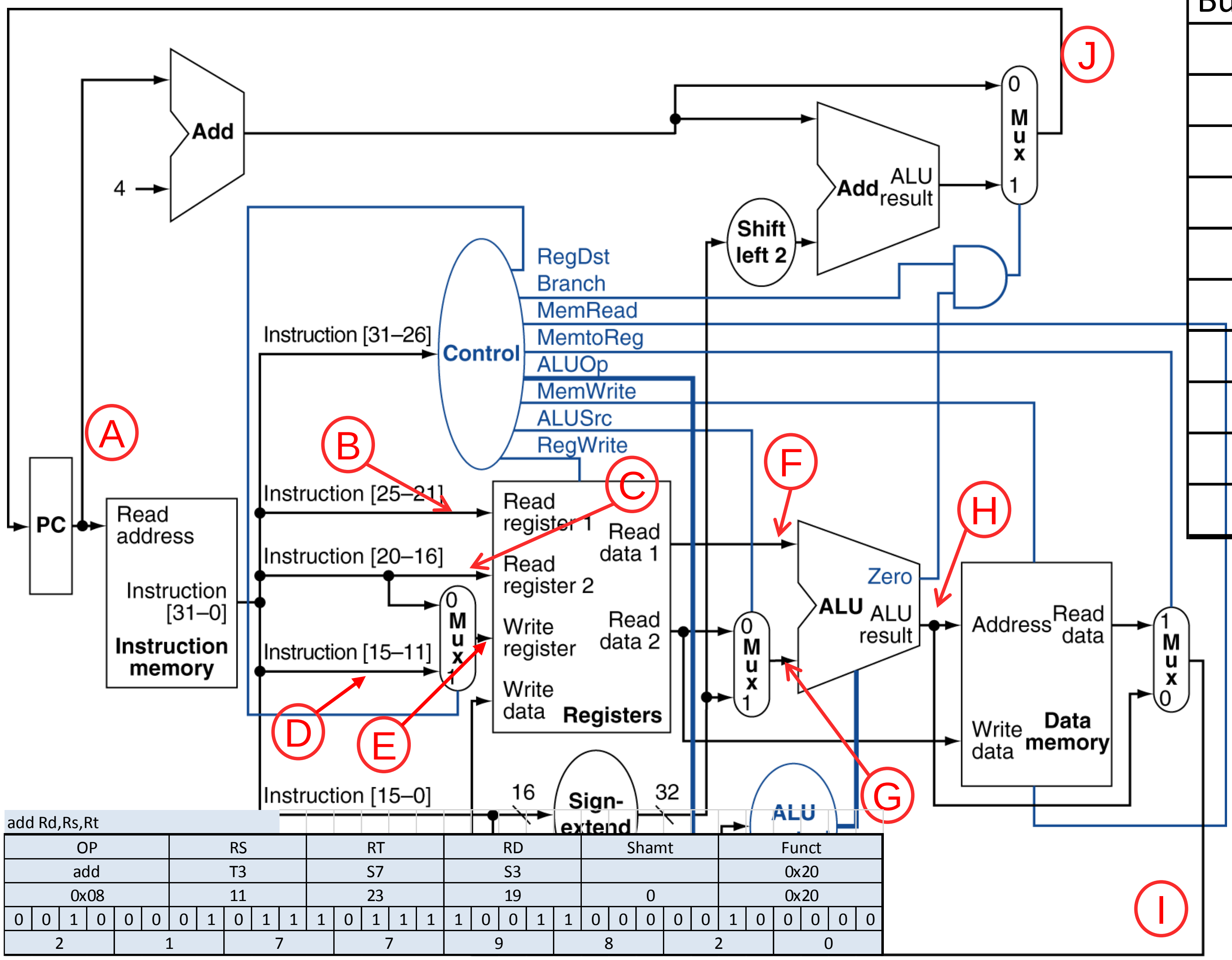
- Performance Issues
  - Longest delay determines clock period
    - Critical path: load instruction
    - Instruction memory → register file → ALU → data memory → register file
  - Not feasible to vary period for different instructions
  - Violates design principle
    - Making the common case fast
  - We will improve performance by pipelining

After completion of the instruction “add $s3,$t3,$s7” indicate the value of each data bus. Assume $t3=0xDCBA, $s7=0x4321, and the instruction was located at memory location 0x1220, use x for unknown



Bus/Wire	Value (hex)
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	

After completion of the instruction "add $s3,$t3,$s7" indicate the value of each data bus. Assume $t3=0xDCBA, $s7=0x4321, and the instruction was located at memory location 0x1220, use x for unknown



Bus/Wire	Value (hex)
A	
B	
C	
D	
E	
F	
G	
H	
I	
J	

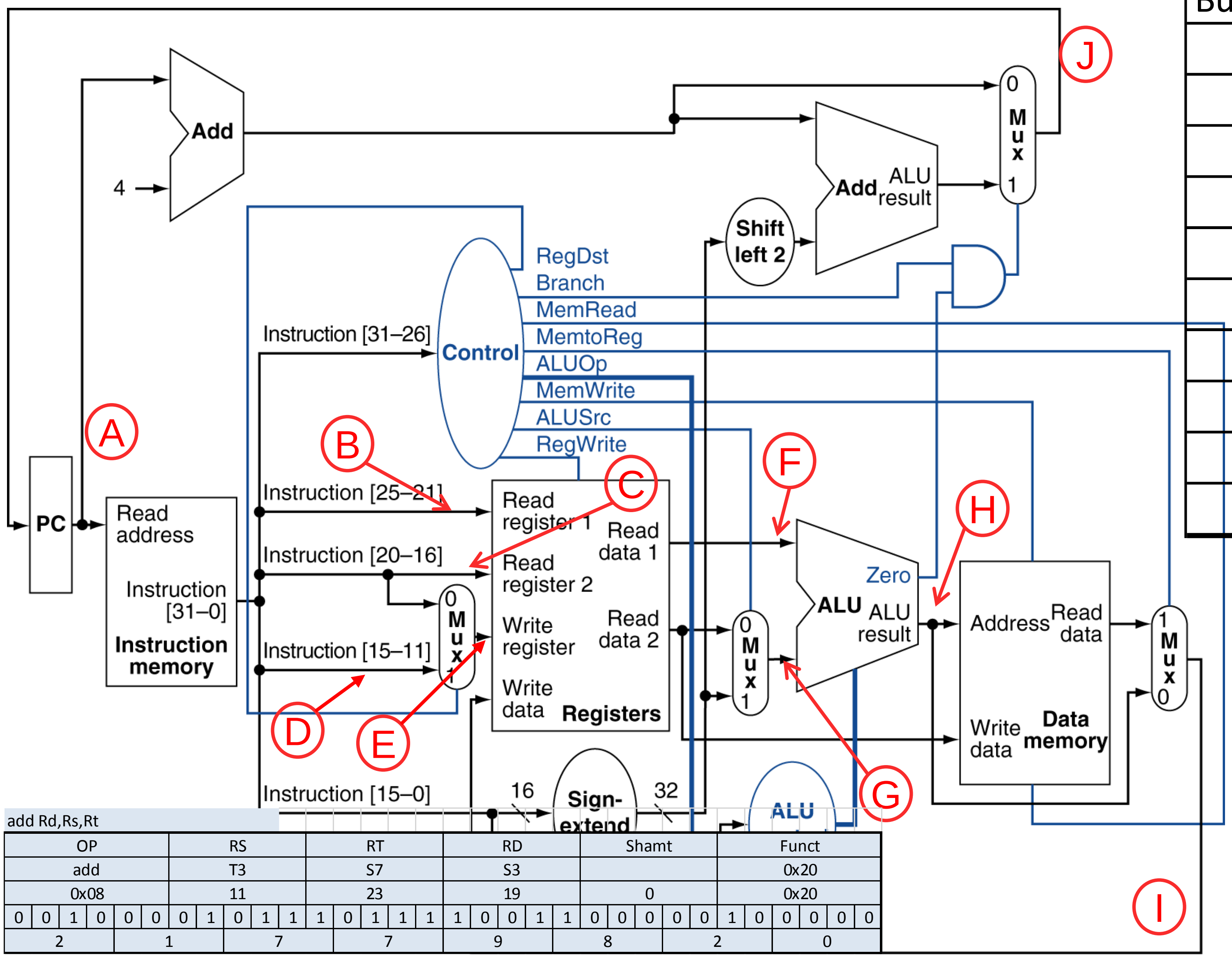
add Rd,Rs,Rt

OP	RS	RT	RD	Shamt	Funct
add	T3	S7	S3		0x20
0x08	11	23	19	0	0x20

0	0	1	0	0	0	0	1	0	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0

2	1	7	7	9	8	2	0

After completion of the instruction "add $s3,$t3,$s7" indicate the value of each data bus. Assume $t3=0xDCBA, $s7=0x4321, and the instruction was located at memory location 0x1220, use x for unknown



Bus/Wire	Value (hex)
A	1220
B	B
C	17
D	13
E	13
F	0000 DCBA
G	0000 4321
H	0001 1FDB
I	0001 1FDB
J	1224

add Rd,Rs,Rt

OP	RS	RT	RD	Shamt	Funct
add	T3	S7	S3		0x20
0x08	11	23	19	0	0x20

0	0	1	0	0	0	0	1	0	1	1	1	0	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	0	0	0	0

2	1	7	7	9	8	2	0