CS2100

TUTORIAL #5
MIPS PROCESSOR: DATAPATH AND CONTROL

(PREPARED BY: AARON TAN)

From lecture slide:

Generating ALUControl Signal

Opcode	ALUop	Instruction Operation	Funct field	ALU action	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	10 0000	add	0010
R-type	10	subtract	10 0010	subtract	0110
R-type	10	AND	10 0100	AND	0000
R-type	10	OR	10 0101	OR	0001
R-type	10	set on less than	10 1010	set on less than	0111

Instruction Type	ALUop
lw / sw	00
beq	01
R-type	10

ALUcontrol	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	slt
1100	NOR

Generation of 2-bit **ALUop** signal will be discussed later

From lecture slide:

Design of ALU Control Unit (1/2)

- Input: 6-bit **`Funct`** field and 2-bit **`ALUop`**
- Output: 4-bit **`ALUcontrol`**
- Find the simplified expressions

ALUcontrol3 = 0

ALUcontrol2 = ?

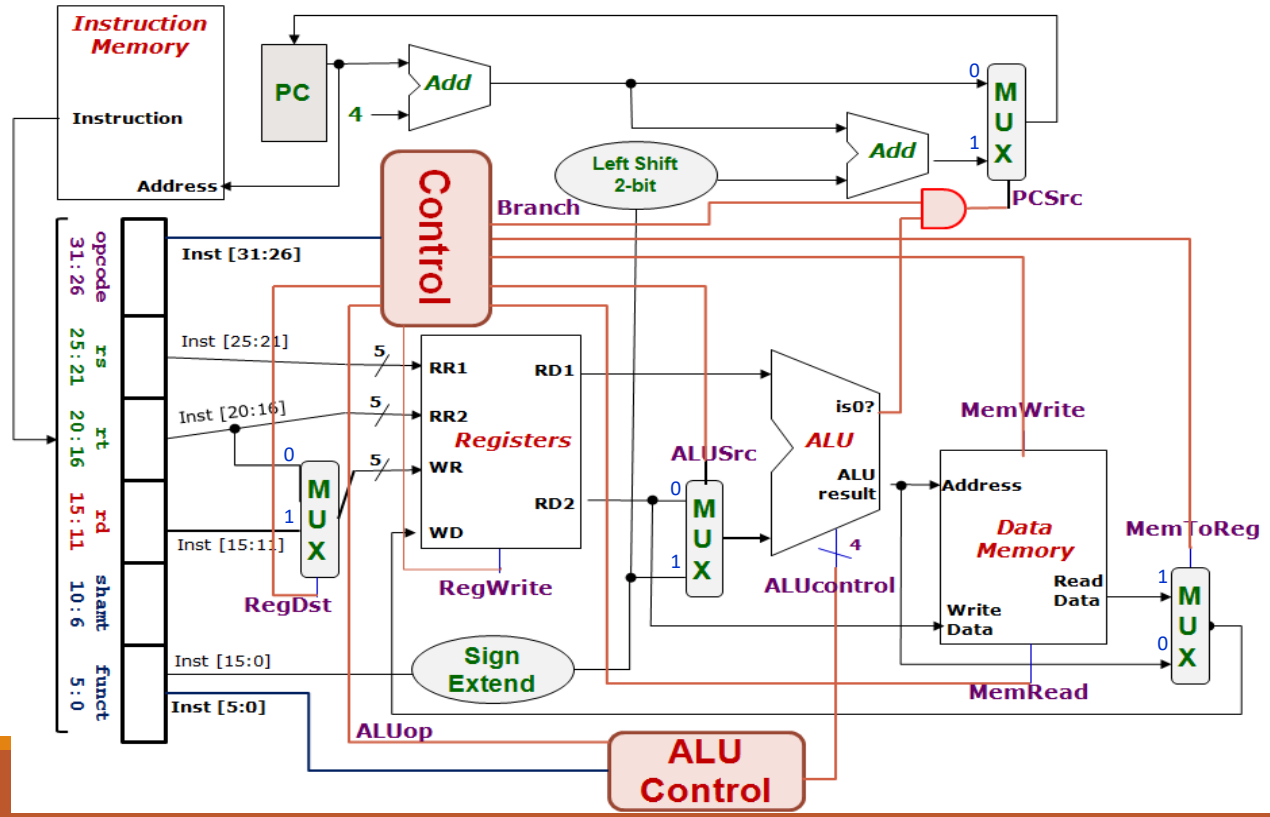
ALUop0 + ALUop1· F1

	ALUop		Funct Field (F[5:0] == Inst[5:0])						ALU control
	MSB	LSB	F5	F4	F3	F2	F1	F0	
lw	0	0	X	X	X	X	X	X	0 0 1 0
sw	0	0	X	X	X	X	X	X	0 0 1 0
beq	~~0~~ X	1	X	X	X	X	X	X	0 1 1 0
add	1	~~0~~ X	~~1~~ X	~~0~~ X	0	0	0	0	0 0 1 0
sub	1	~~0~~ X	~~1~~ X	~~0~~ X	0	0	1	0	0 1 1 0
and	1	~~0~~ X	~~1~~ X	~~0~~ X	0	1	0	0	0 0 0 0
or	1	~~0~~ X	~~1~~ X	~~0~~ X	0	1	0	1	0 0 0 1
slt	1	~~0~~ X	~~1~~ X	~~0~~ X	1	0	1	0	0 1 1 1

From lecture slide:

Control Design: **Outputs**

	RegDst	ALUSrc	MemTo Reg	Reg Write	Mem Read	Mem Write	Branch	ALUop	
								op1	op0
R-type	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



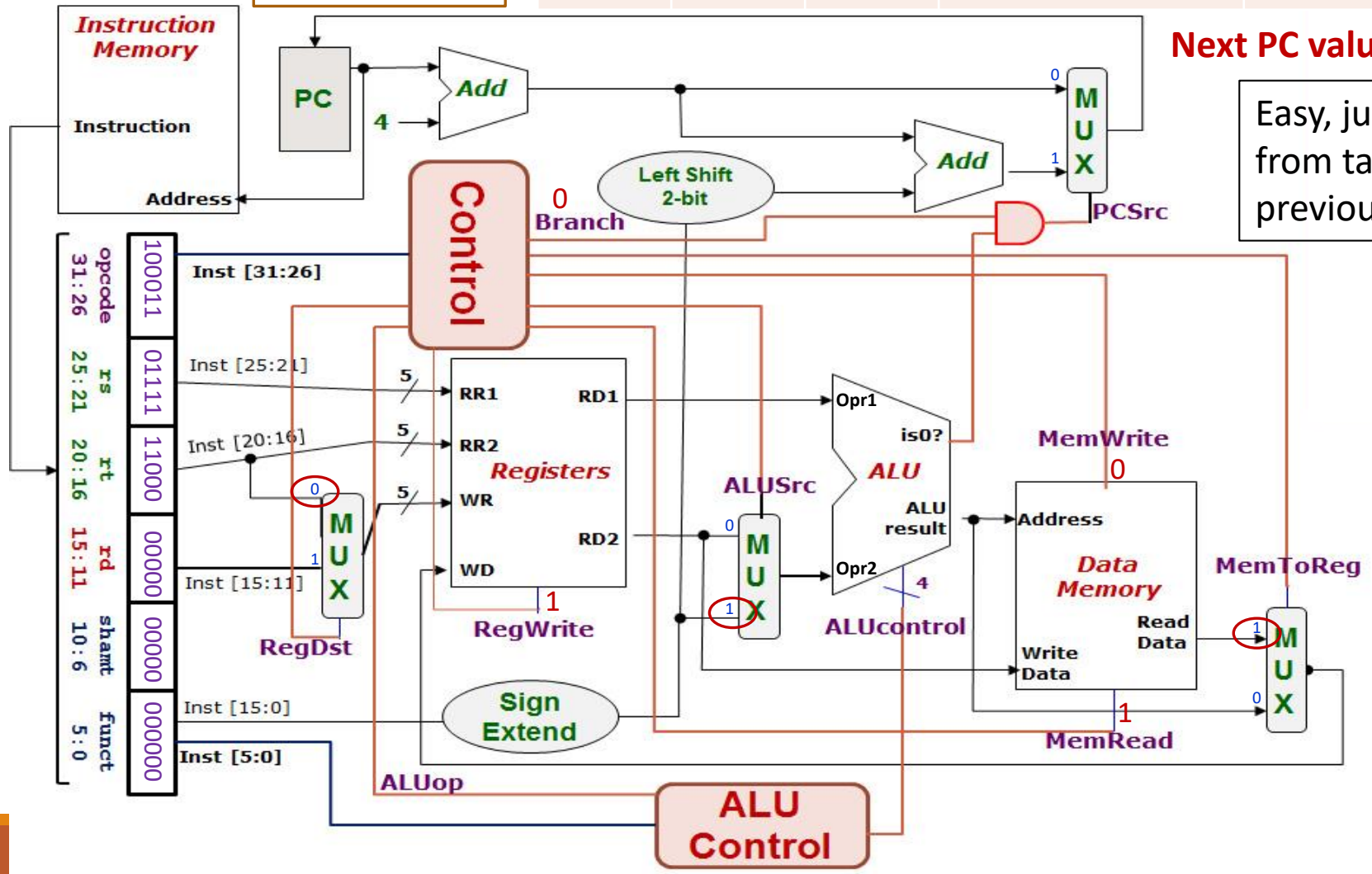
`lw $24, 0($15)` | R[rt] = M[R[rs]+SignExtImm] | `100011 01111 11000 0000000000000000`

Q1(i)

Red means the data is actually used.

Blue means the data is not used.

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$15	$24	$24	MEM([$15]+0)	[$15]	0	[$15]+0	[$24]



Next PC value = PC + 4

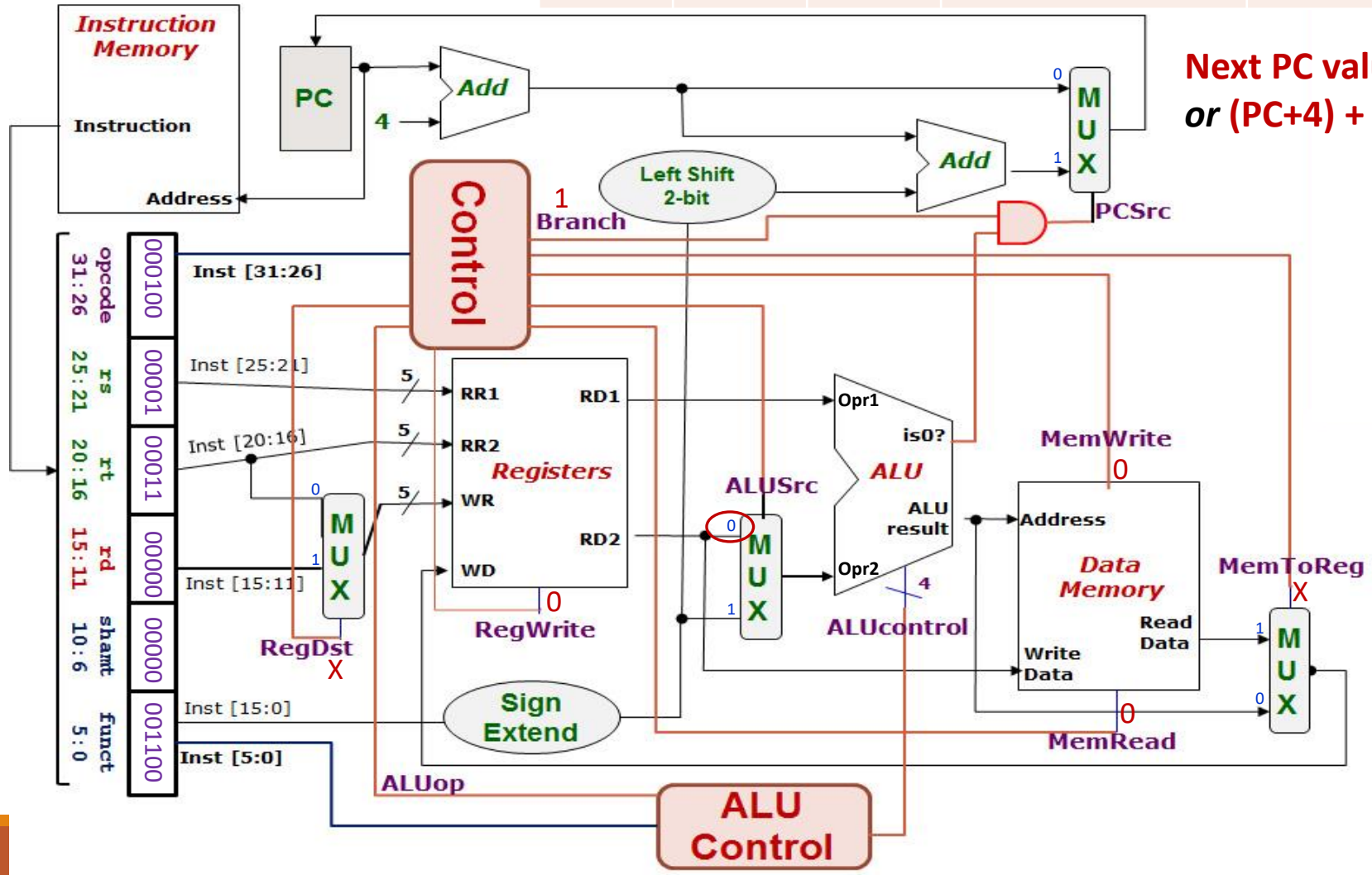
Easy, just copy from tables in previous slides.

RegDest	0
RegWrite	1
ALUSrc	1
MemRead	1
MemWrite	0
MemToReg	1
Branch	0
ALUop	00
ALUcontrol	0010

`beq $1, $3, 12` | If (R[rs]==R[rt]) PC=PC+4+BrAddr | `000100 00001 00011 0000000000001100`

Q1(ii)

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$1	**$3**	$3 or $0	[$1]-[$3] or random value	**[$1]**	**[$3]**	[$1] – [$3]	[$3]



Next PC value = PC + 4
***or* (PC+4) + (12×4)**

RegDest	**X**
RegWrite	**0**
ALUSrc	**0**
MemRead	**0**
MemWrite	**0**
MemToReg	**X**
Branch	**1**
ALUop	**01**
ALUcontrol	**0110**

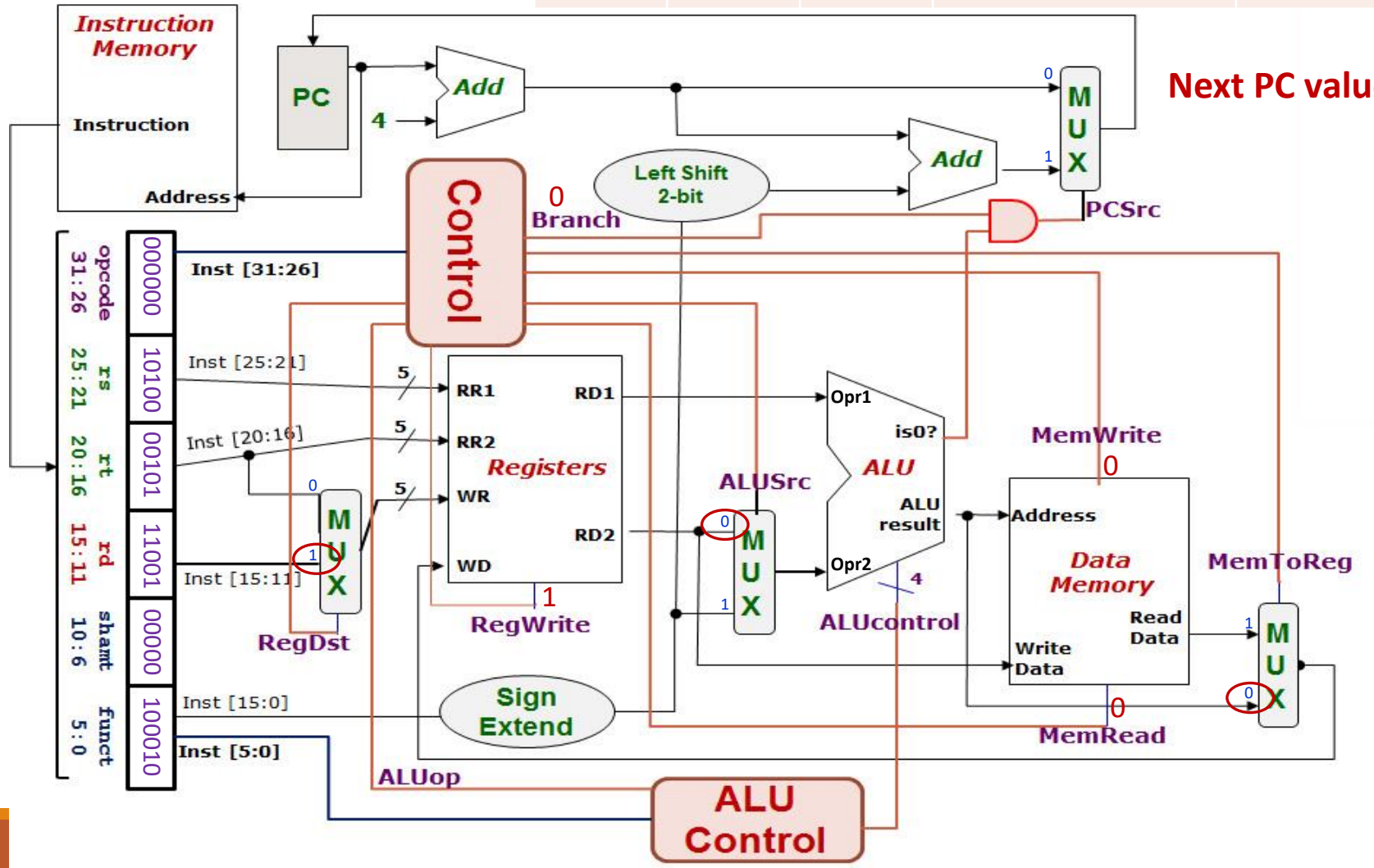
`sub $25, $20, $5`

R[rd] = R[rs] – R[rt]

`000000 10100 00101 11001 00000 100010`

Q1(iii)

Registers File				ALU		Data Memory	
RR1	RR2	WR	WD	Opr1	Opr2	Address	Write Data
$20	**$5**	**$25**	**[$20] – [$5]**	**[$20]**	**[$5]**	[$20] – [$5]	[$5]



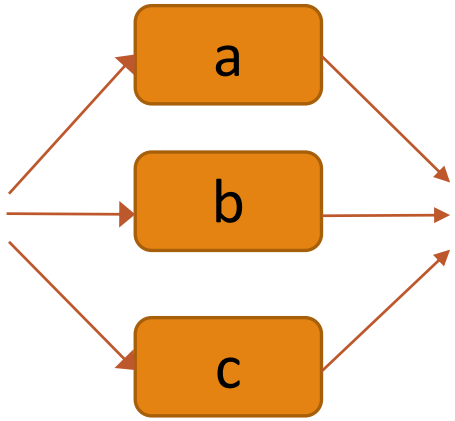
Next PC value = PC + 4

RegDest	**1**
RegWrite	**1**
ALUSrc	**0**
MemRead	**0**
MemWrite	**0**
MemToReg	**0**
Branch	**0**
ALUop	**10**
ALUcontrol	**0110**

Critical Path Analysis

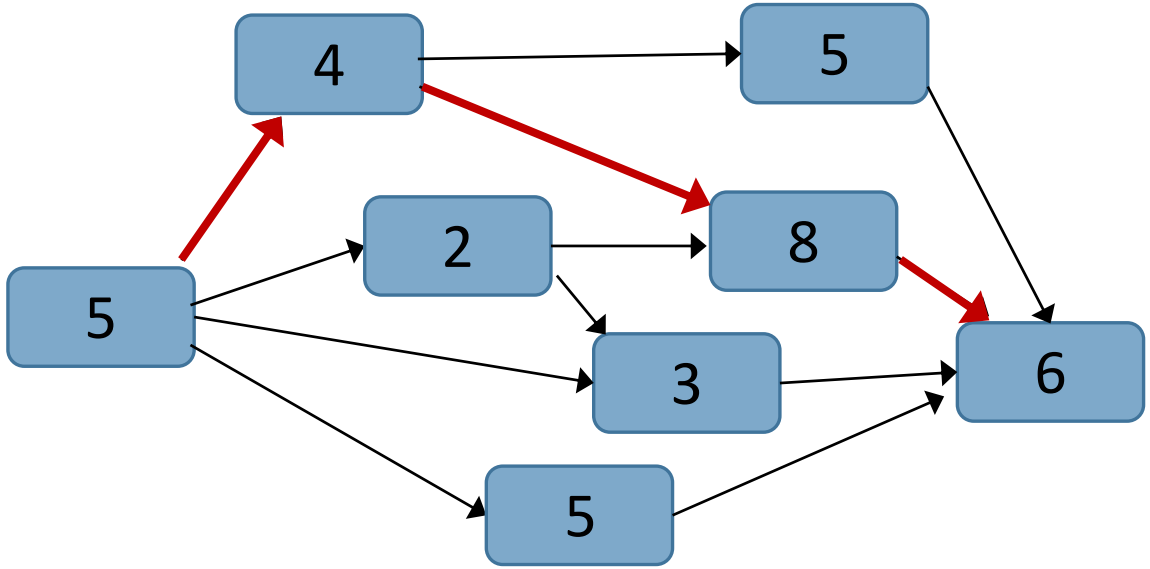
Duration = a + b + c

Sequence/Series



Duration = max(a,b,c)

Parallel



Critical path duration = 5 + 4 + 8 + 6 = 23

SUB instruction

Inst-Mem (400) → Reg.File (200) → MUX (ALUSrc) (30) → ALU (120) → MUX (MToR) (30) → Reg.File (200)

Control (100)

Not critical path

Q2(a)

Inst-Mem
400ps

Adder
100ps

MUX
30ps

ALU
120ps

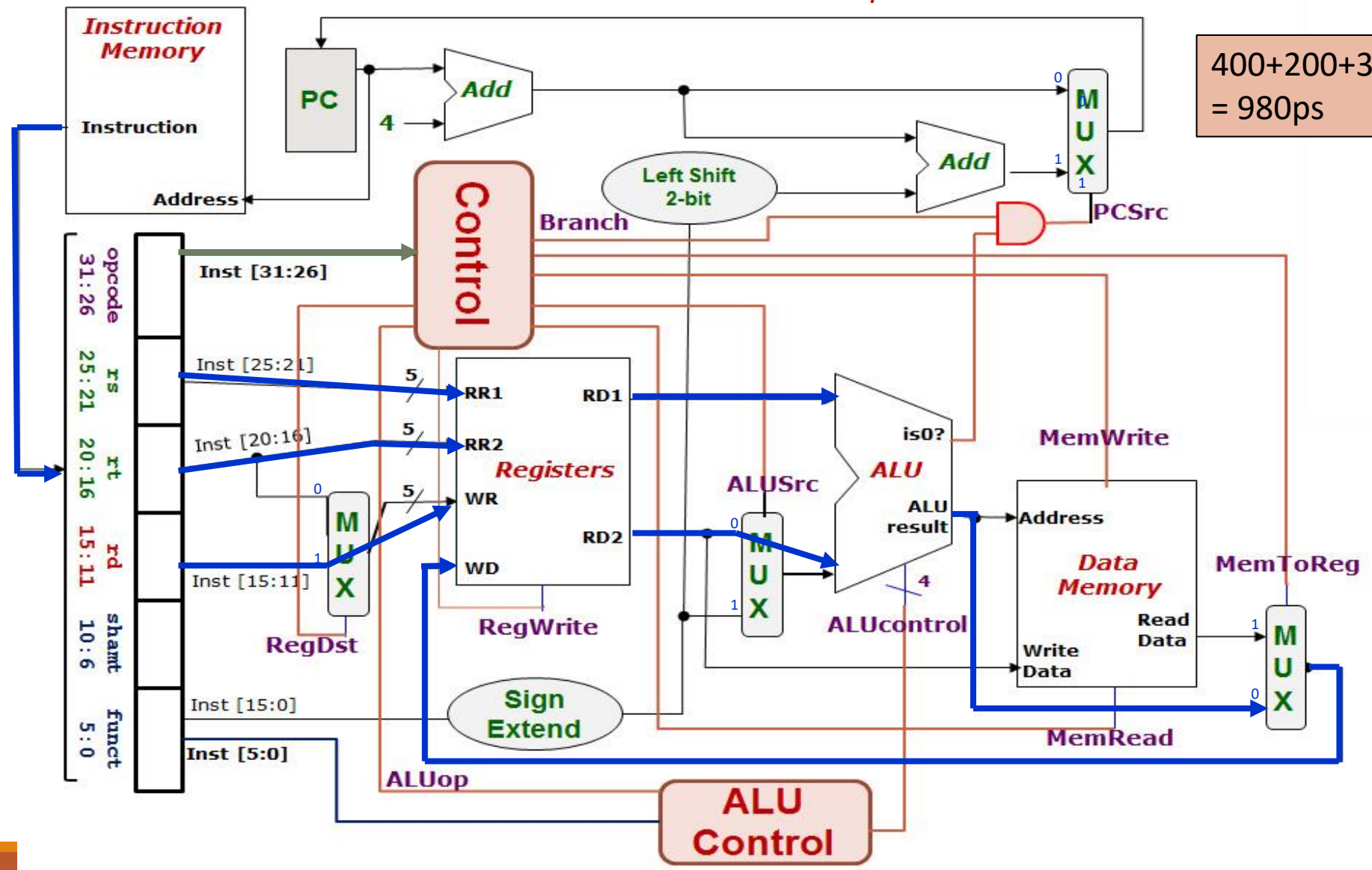
Reg-File
200ps

Data-Mem
350ps

Control/ALU Control
100ps

Lshft/signext /AND
20ps

400+200+30+120+30+200 = 980ps



LW instruction

Inst-Mem (400) → Reg.File (200) → ALU (120) → DataMem (350) → MUX (MToR) (30) → Reg.File (200)

Control (100)

Not critical path

Q2(b)

Inst-Mem
400ps

Adder
100ps

MUX
30ps

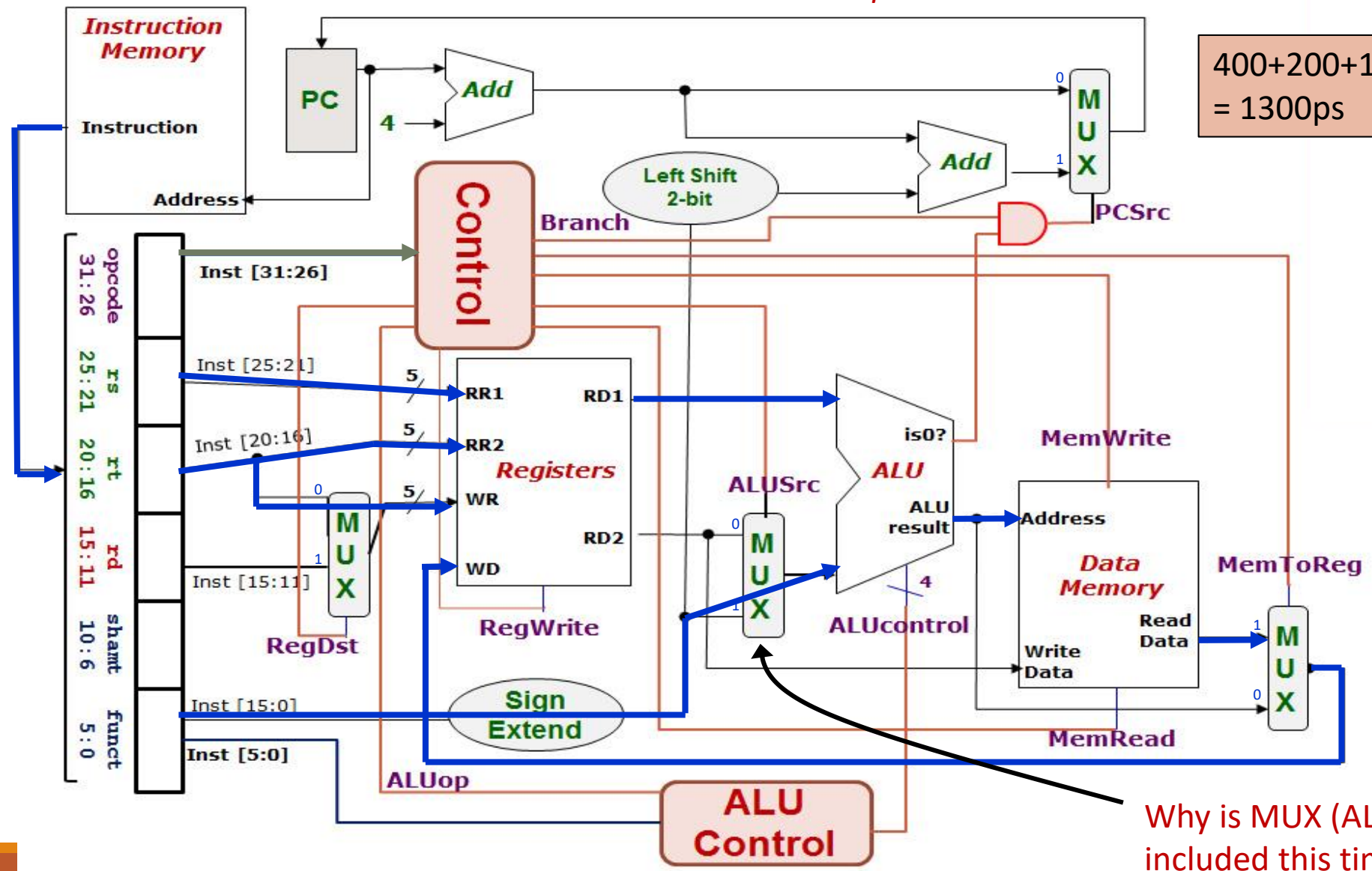
ALU
120ps

Reg-File
200ps

Data-Mem
350ps

Control/ALU Control
100ps

Lshft/signext /AND
20ps



400+200+120+350+30+200 = 1300ps

Why is MUX (ALUSrc) not included this time?

BEQ instruction

Inst-Mem (400) → Reg.File (200) → MUX (ALUSrc) (30) → ALU (120) → AND (20) → MUX (PCSrc) (30)

Control (100)

Not critical path

Q2(c)

Inst-Mem
400ps

Adder
100ps

MUX
30ps

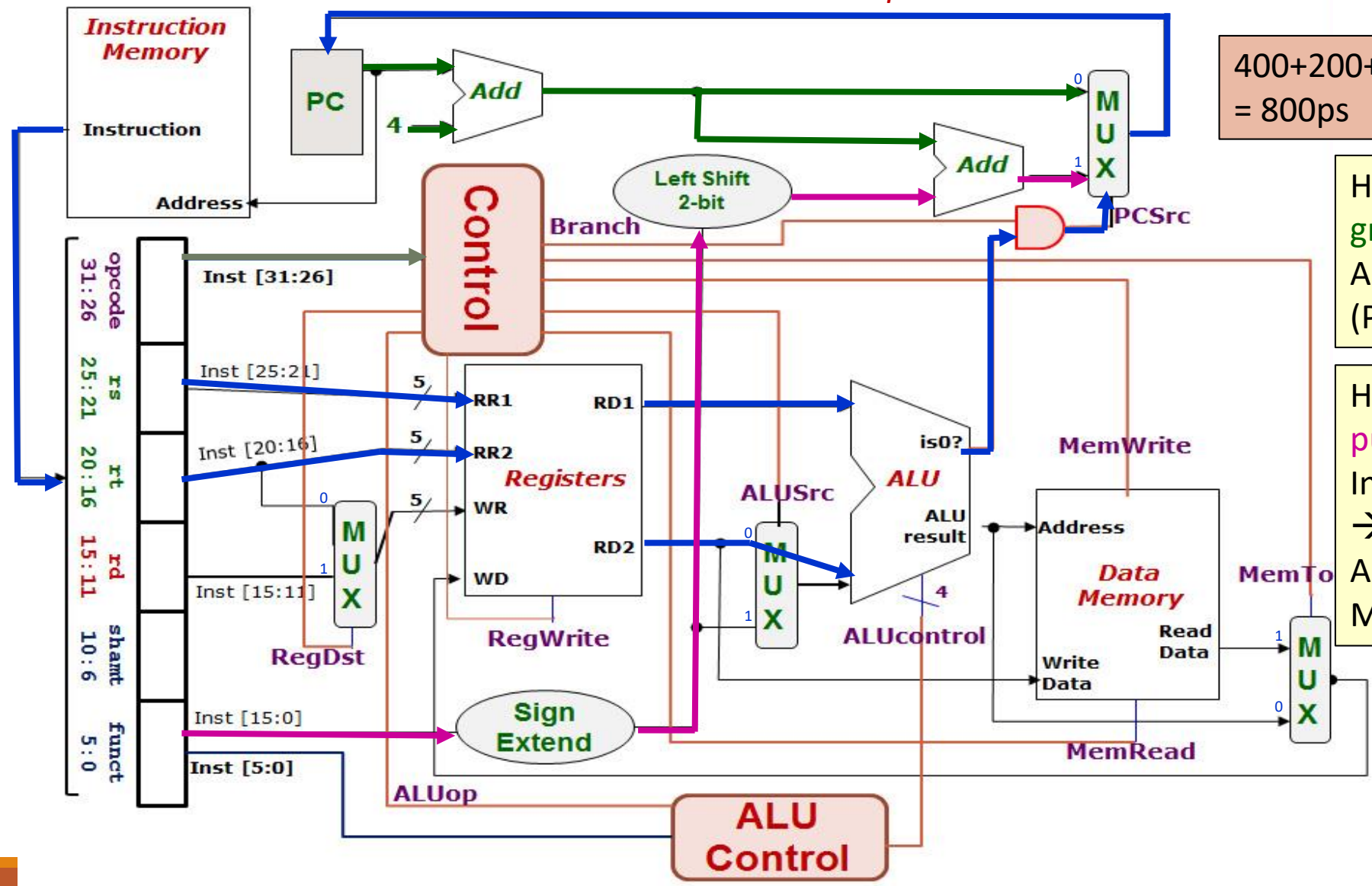
ALU
120ps

Reg-File
200ps

Data-Mem
350ps

Control/ALU Control
100ps

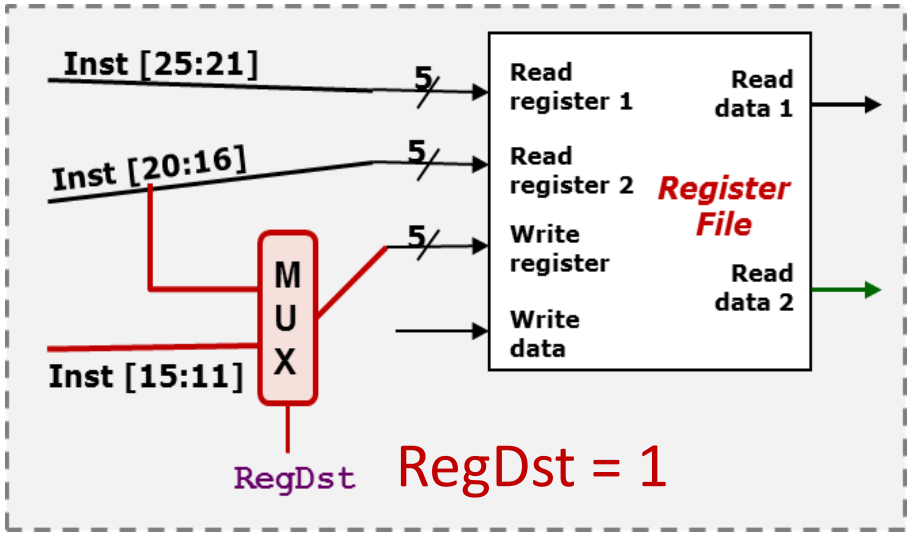
Lshft/signext /AND
20ps

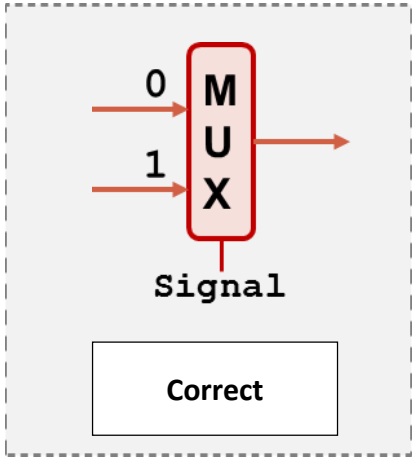


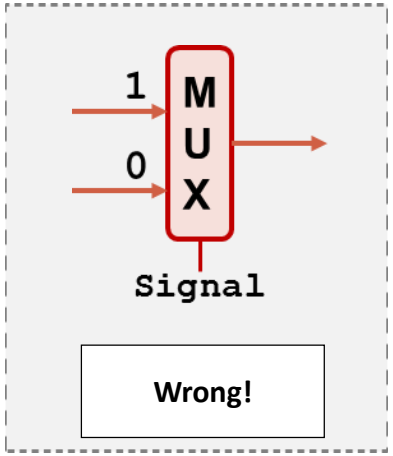
400+200+30+120+20+30 = 800ps

How about the green path: PC → Adder → MUX (PCSrc)?

How about the purple path: Inst.Mem → SignExt → LeftShift → Adder → MUX(PCSrc)?







R[rd] = R[rs] + R[rt]

opcode	rs	rt	rd	shamt	funct

Q3(a) add instruction

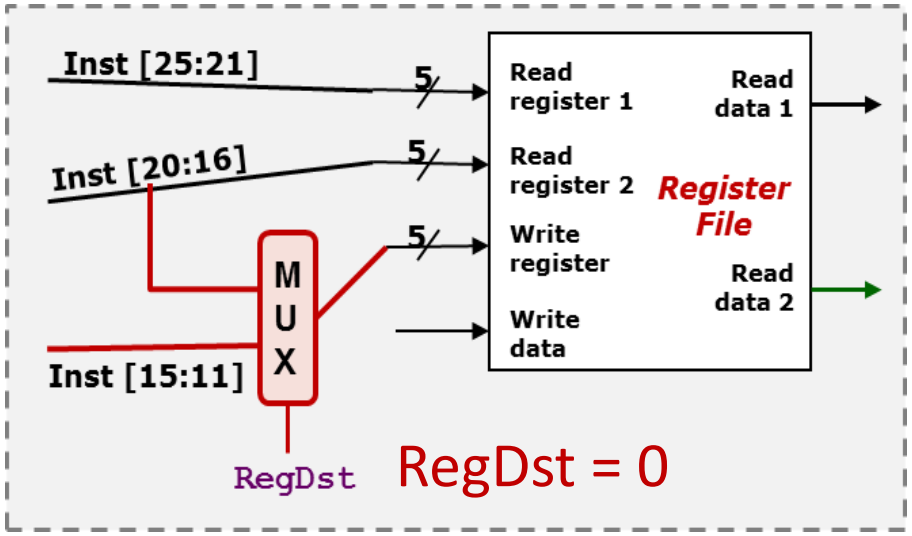
(i) One example where the incorrect processor still gives the right execution result.

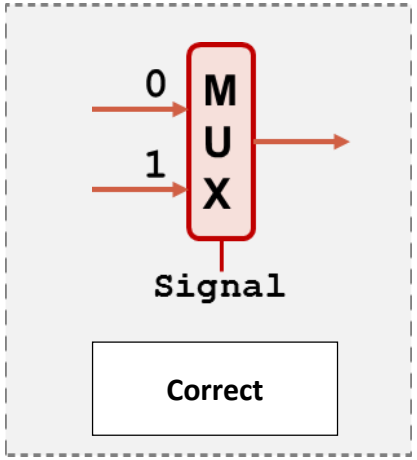
Many possible answers.
Make RT = RD.

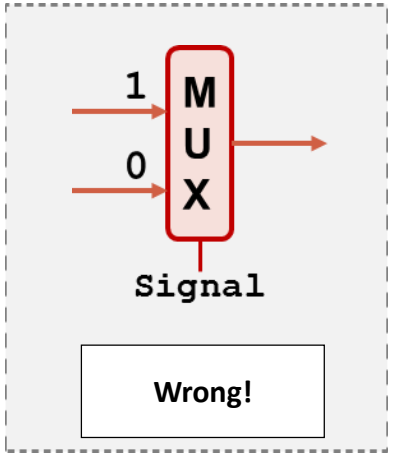
Example: add $t0, $t1, $t0

(ii) One example where the incorrect processor gives the wrong execution result.

Example: add $t0, $t1, $t2
$t2 instead of $t0 is picked as write register.







R[rt] = M[R[rs] + SignExtImm]

opcode	rs	rt	immed

Q3(b) lw instruction

(i) One example where the incorrect processor still gives the right execution result.

Make the first 5 bits of immediate value the same as the register number of RT.

Example: lw $a0, 8192($t0)

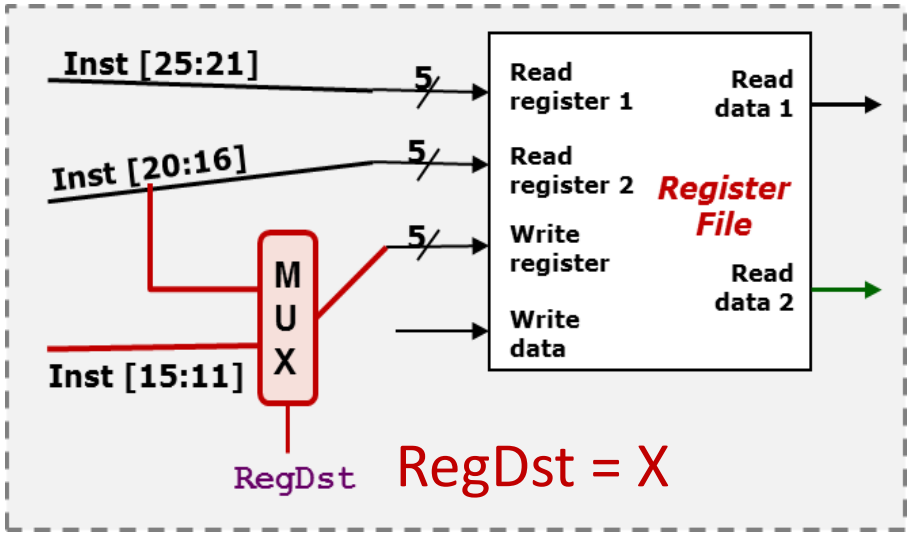
8192 = 0x2000

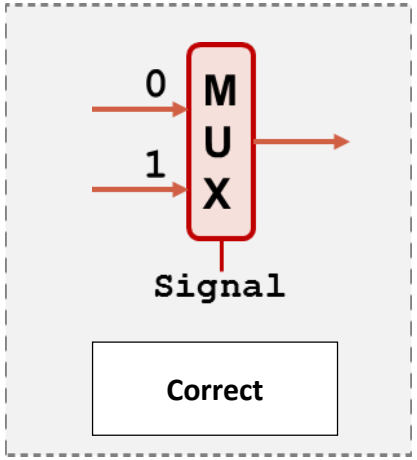
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

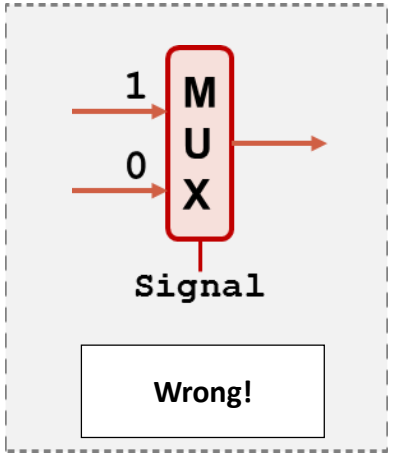
$a0 = $4

(ii) One example where the incorrect processor gives the wrong execution result.

Anything other than (i).







Q3(c) beq instruction

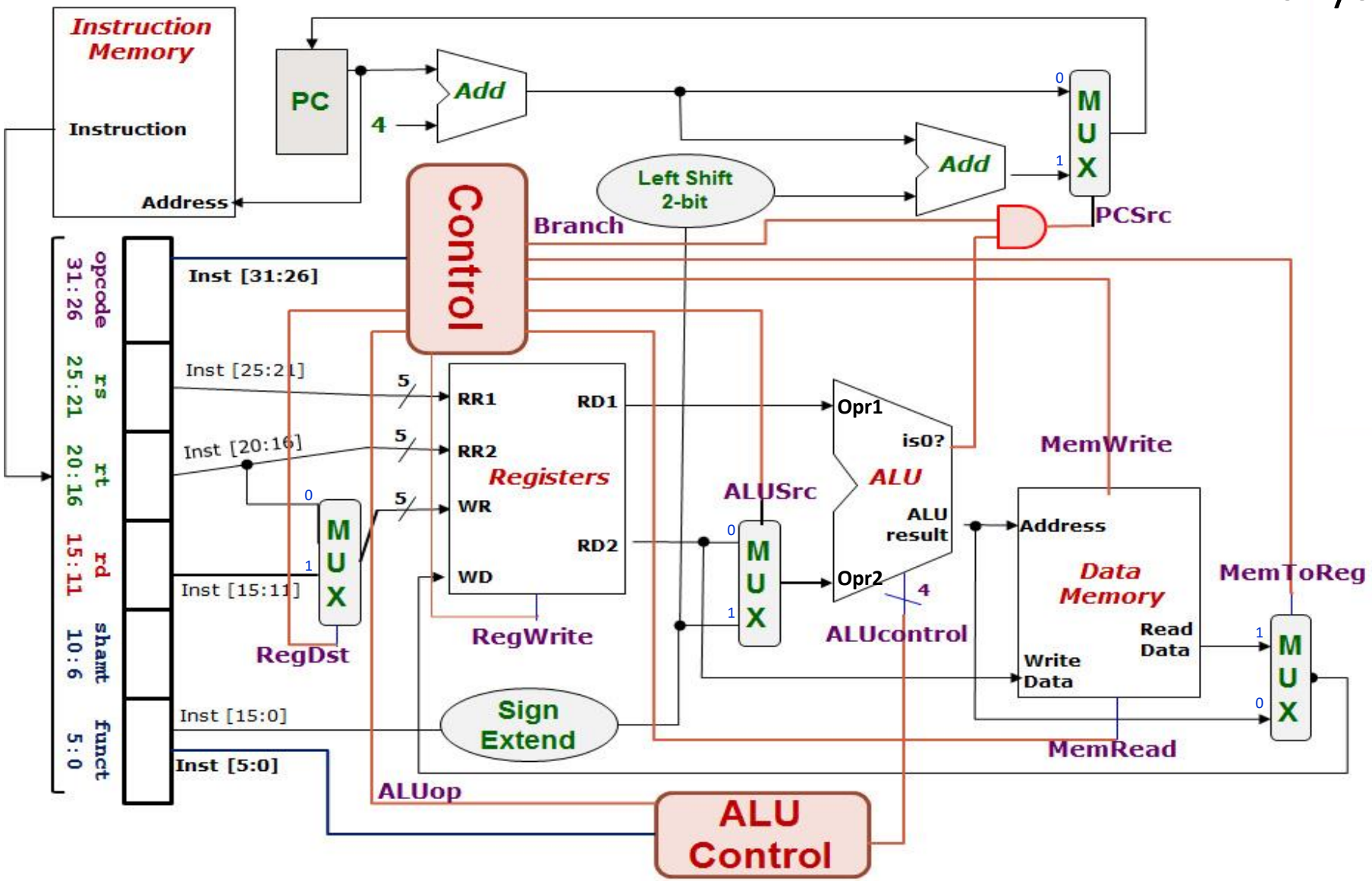
(i) One example where the incorrect processor still gives the right execution result.

Anything will work, since error has no impact on branch instructions.

(ii) One example where the incorrect processor gives the wrong execution result.

None.

For your use.


Instruction Memory
Instruction
Address
PC
4
Add
Add
Left Shift 2-bit
0
1
M U X
PCSrc
Branch
Control
opcode 31:26
rs 25:21
rt 20:16
rd 15:11
shamt 10:6
funct 5:0
Inst [31:26]
Inst [25:21]
Inst [20:16]
Inst [15:11]
Inst [15:0]
Inst [5:0]
5
RR1
RR2
WR
WD
RD1
RD2
Registers
RegDst
RegWrite
ALUSrc
Opr1
Opr2
is0?
ALU
ALU result
4
ALUcontrol
MemWrite
Address
Data Memory
Read Data
Write Data
MemRead
MemToReg
Sign Extend
ALUop
ALU Control

END OF FILE