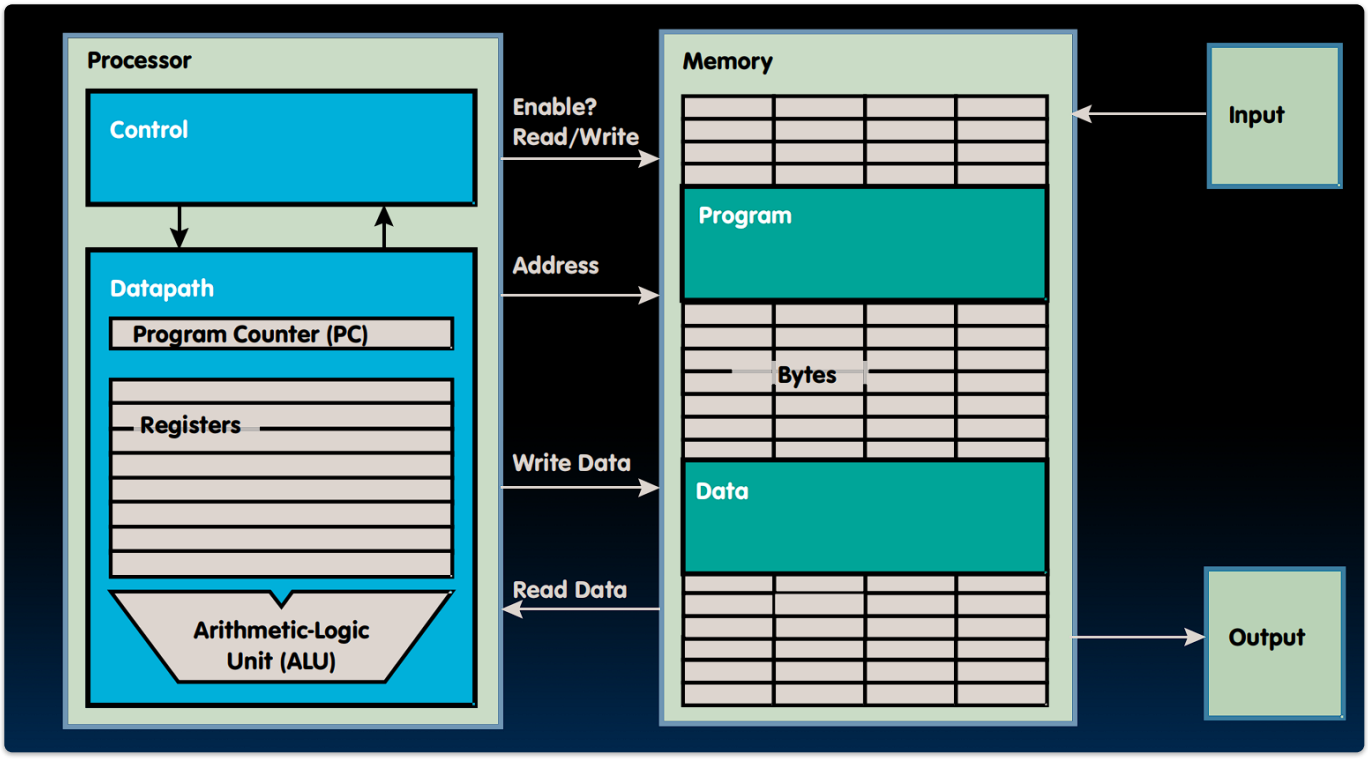
Single-Core Processor



CPU

The active part of the computer that does all the work.

- Data manipulation.
- Decision making.

Datapath

Portion of the processor that contains hardware necessary to perform operations required by the processor(the body)

Control

Portion of the processor that tells the datapath what needs to be done.(the brain)

One-Instruction-Per-Cycle RISC-V Machine

Stage of the Datapath

break up the process of "executing an instruction" into stages, and then connect the stages to create the whole datapath.

- **Stage 1: *Instruction Fetch (IF)***
- **Stage 2: *Instruction Decode (ID)***
- **Stage 3: *Execute (EX) - ALU* (Arithmetic-Logic Unit)**
- **Stage 4: *Memory Access (MEM)***
- **Stage 5: *Write Back to Register (WB)***

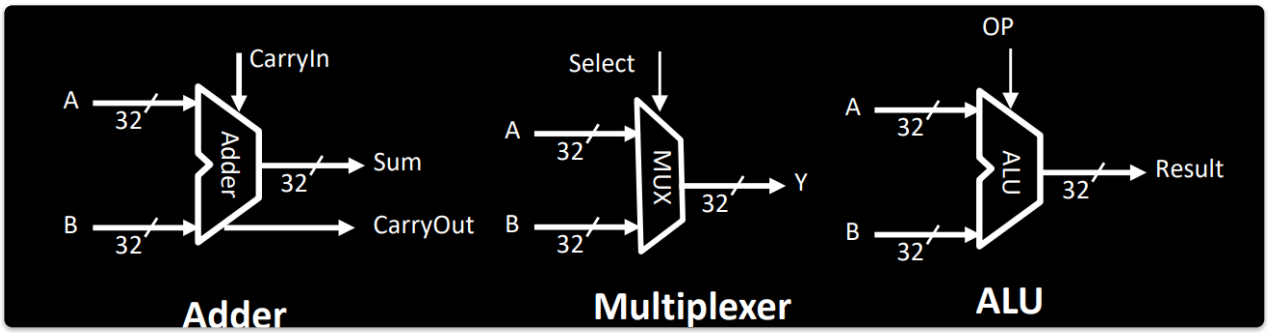
These five stages are executed in one clock cycle, which is called One-instruction-Per-cycle.

Register write will be executed at the next rising edge of clock.

Datapath components

COMBINATIONAL

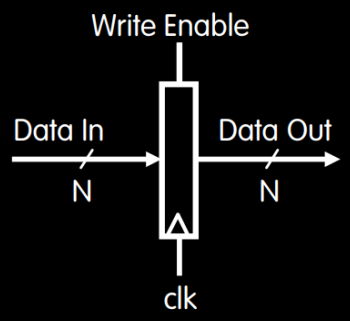
- Combinational elements



STATE AND SEQUENCING

Register

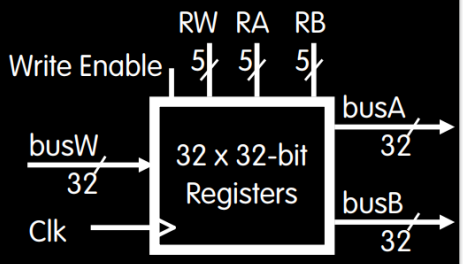
- Register
- Write Enable:
  - Low (or deasserted) (0): Data Out will not change
  - Asserted (1): Data Out will become Data In on positive edge of clock



Register File

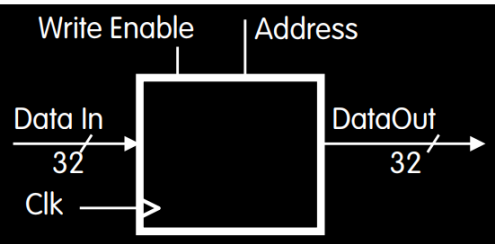
Consists of 32 registers

- Register file (regfile, RF) consists of 32 registers:
  - Two 32-bit output busses: busA and busB
  - One 32-bit input bus: busW
- Register is selected by:
  - RA (number) selects the register to put on busA (data)
  - RB (number) selects the register to put on busB (data)
  - RW (number) selects the register to be written via busW (data) when Write Enable is 1
- Clock input (Clk)
  - Clk input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block:
    - RA or RB valid ⇒ busA or busB valid after "access time."



Memory

- "Magic" Memory
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - For Read: Address selects the word to put on Data Out
  - For Write: Set Write Enable = 1: address selects the memory word to be written via the Data In bus
- Clock input (CLK)
  - CLK input is a factor ONLY during write operation
  - During read operation, behaves as a combinational logic block: Address valid ⇒ Data Out valid after "access time"



For **read** operation we **don't** need to wait for the clock, just put the address of register or memory, the data will automatically pop up.
For **write** operation we need to wait for the rising edge of the clock to write data.
Each instruction during execution reads and updates the state of

1. registers
2. pc
3. Memory

Datapath

R-Format Datapath

Review

31 30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15	14 13 12	11 10 9 8 7	6 5 4 3 2 1 0
R-format : ALU					
[31:25]	[24:20]	[19:15]	[14:12]	[11:7]	[6:0]
7	5	5	3	5	7
func7	rs2	rs1	func3	rd	opcode
0000000	rs2	rs1	000 : ADD	rd	0110011:OP-R
0100000	rs2	rs1	000 : SUB	rd	0110011:OP-R
0000000	rs2	rs1	001 : SLL	rd	0110011:OP-R
0000000	rs2	rs1	010 : SLT	rd	0110011:OP-R
0000000	rs2	rs1	011 : SLTU	rd	0110011:OP-R
0000000	rs2	rs1	100 : XOR	rd	0110011:OP-R
0000000	rs2	rs1	101 : SRL	rd	0110011:OP-R
0100000	rs2	rs1	101 : SRA	rd	0110011:OP-R
0000000	rs2	rs1	110 : OR	rd	0110011:OP-R
0000000	rs2	rs1	111 : AND	rd	0110011:OP-R

- E.g. Addition/subtraction `add rd, rs1, rs2`

  `R[rd] = R[rs1] + R[rs2]`

  `sub rd, rs1, rs2`

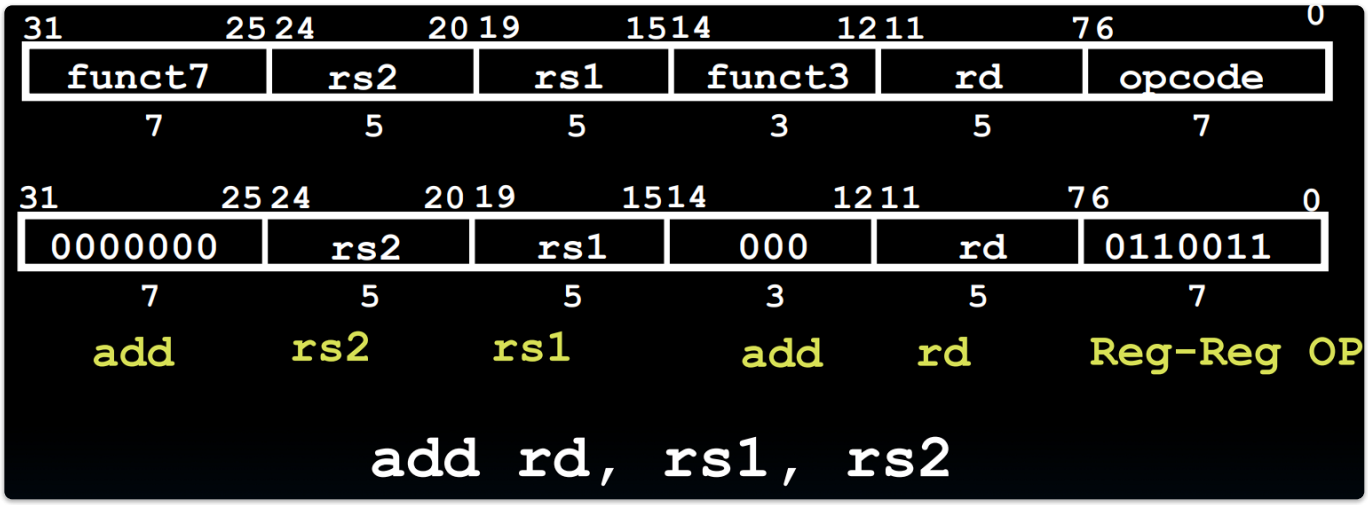
  `R[rd] = R[rs1] - R[rs2]`

Garcia, Nikol

Berkeley UNIVERSITY OF CALIFORNIA

RISC-V (20)

Implementing the add instruction

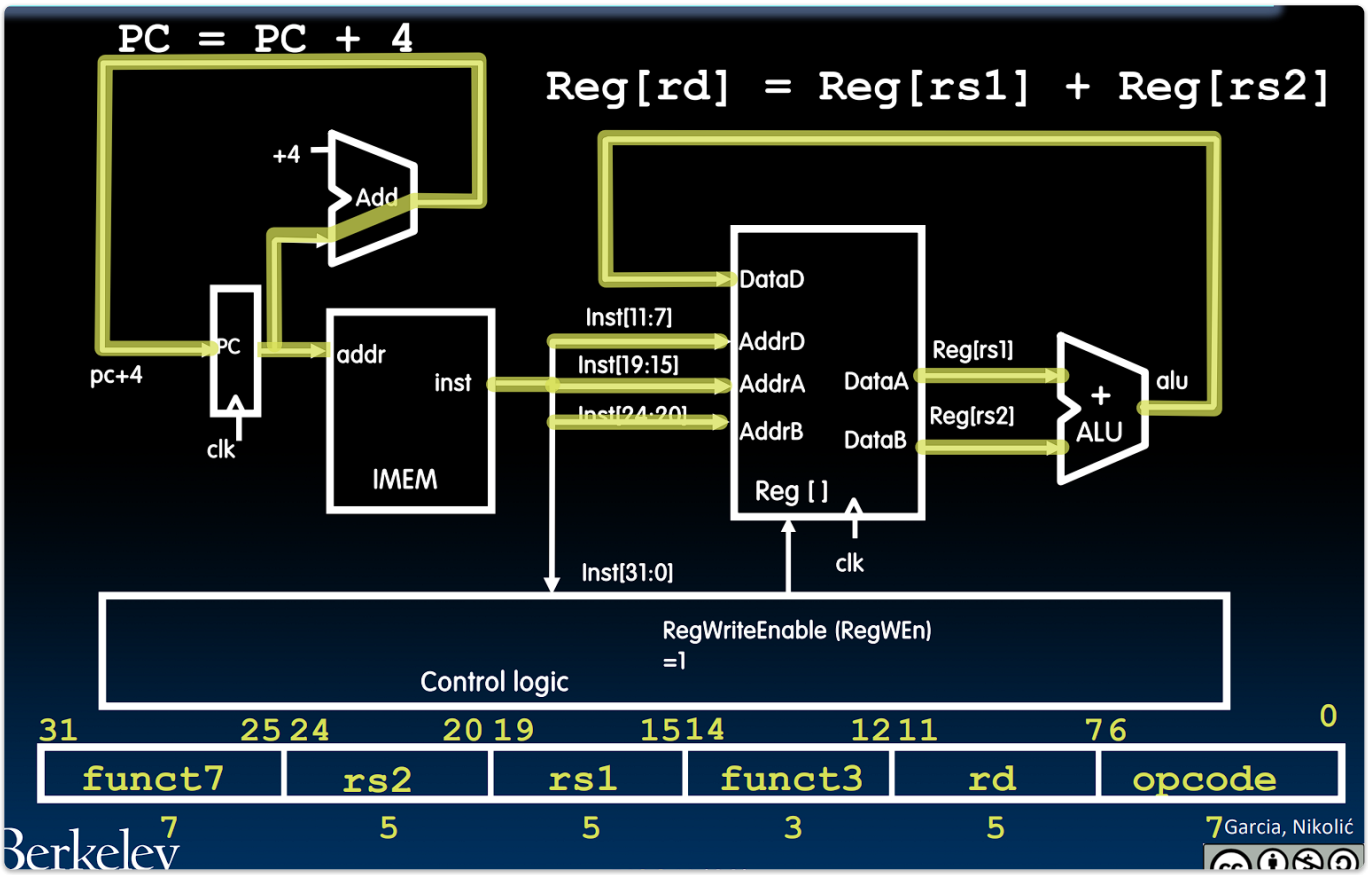


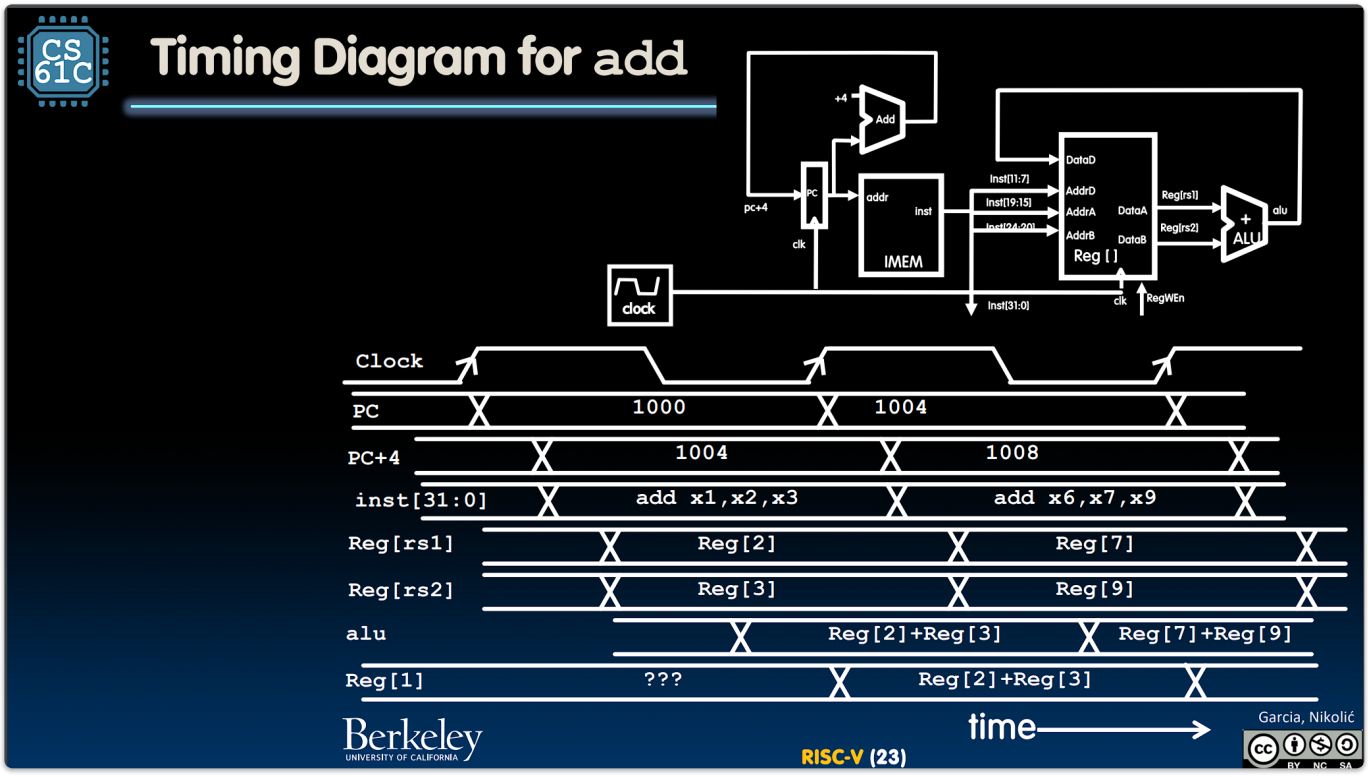
Instruction does two changes

1. **Reg[rd] = Reg[rs1] + Reg[rs2]**

2. PC +=4

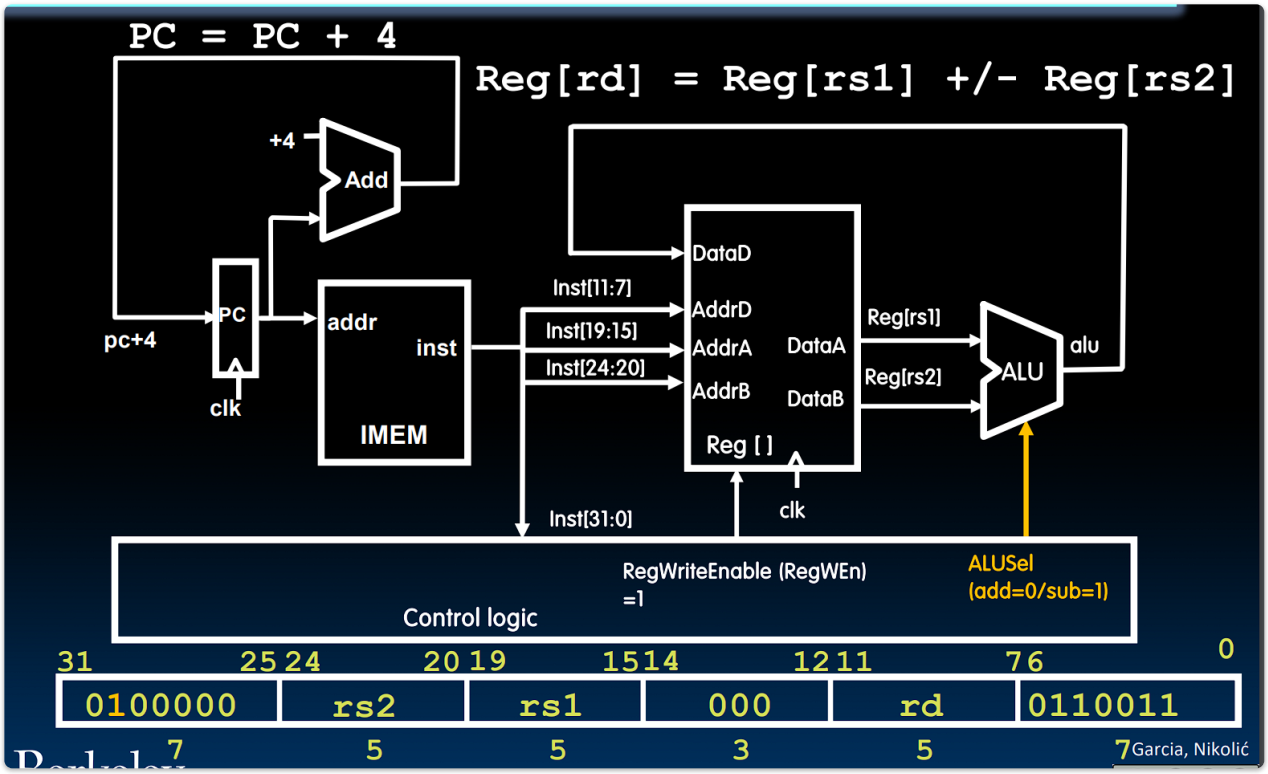
DATAPATH FOR ADD





DATAPATH FOR SUB/ADD

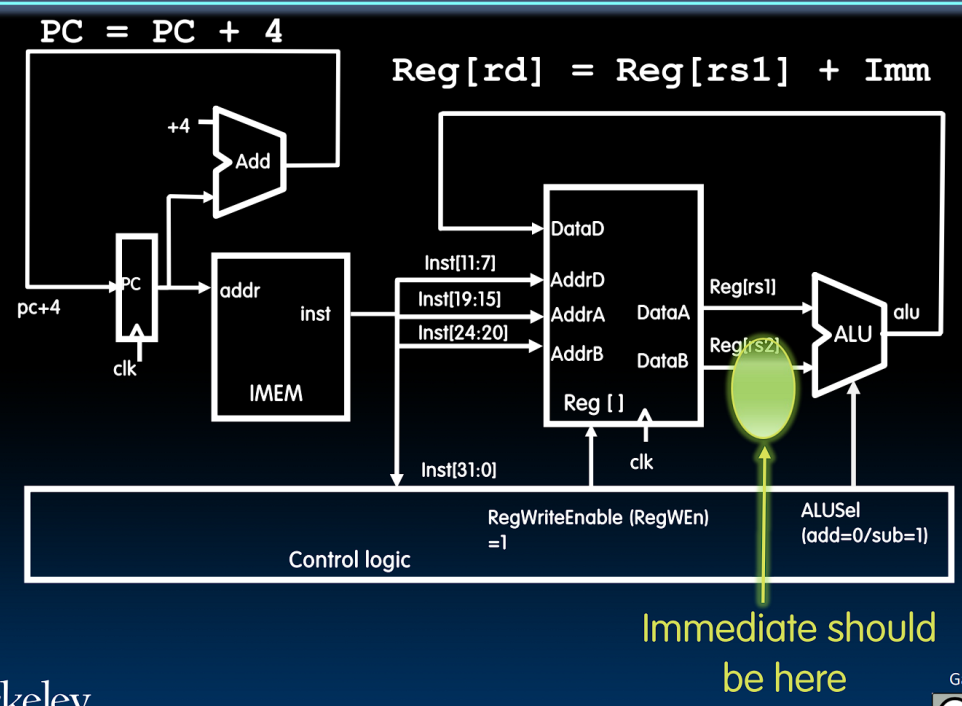
- sub almost the same as add, except now we need to subtract operands.



I-Format Datapath

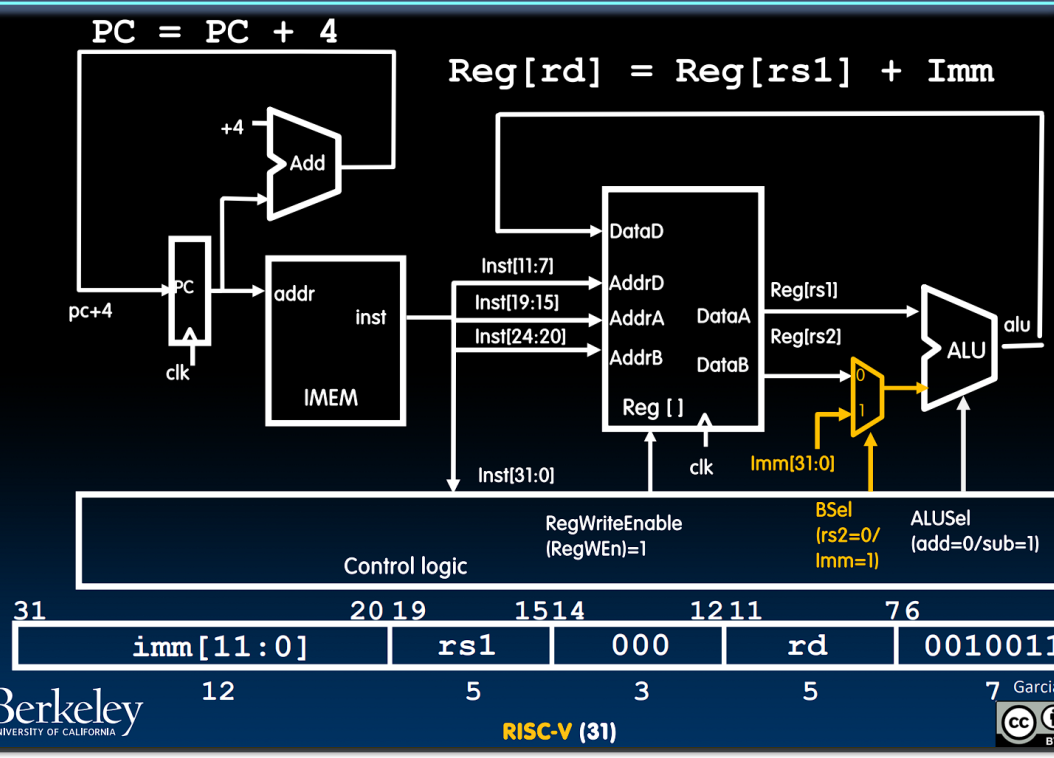
Addi

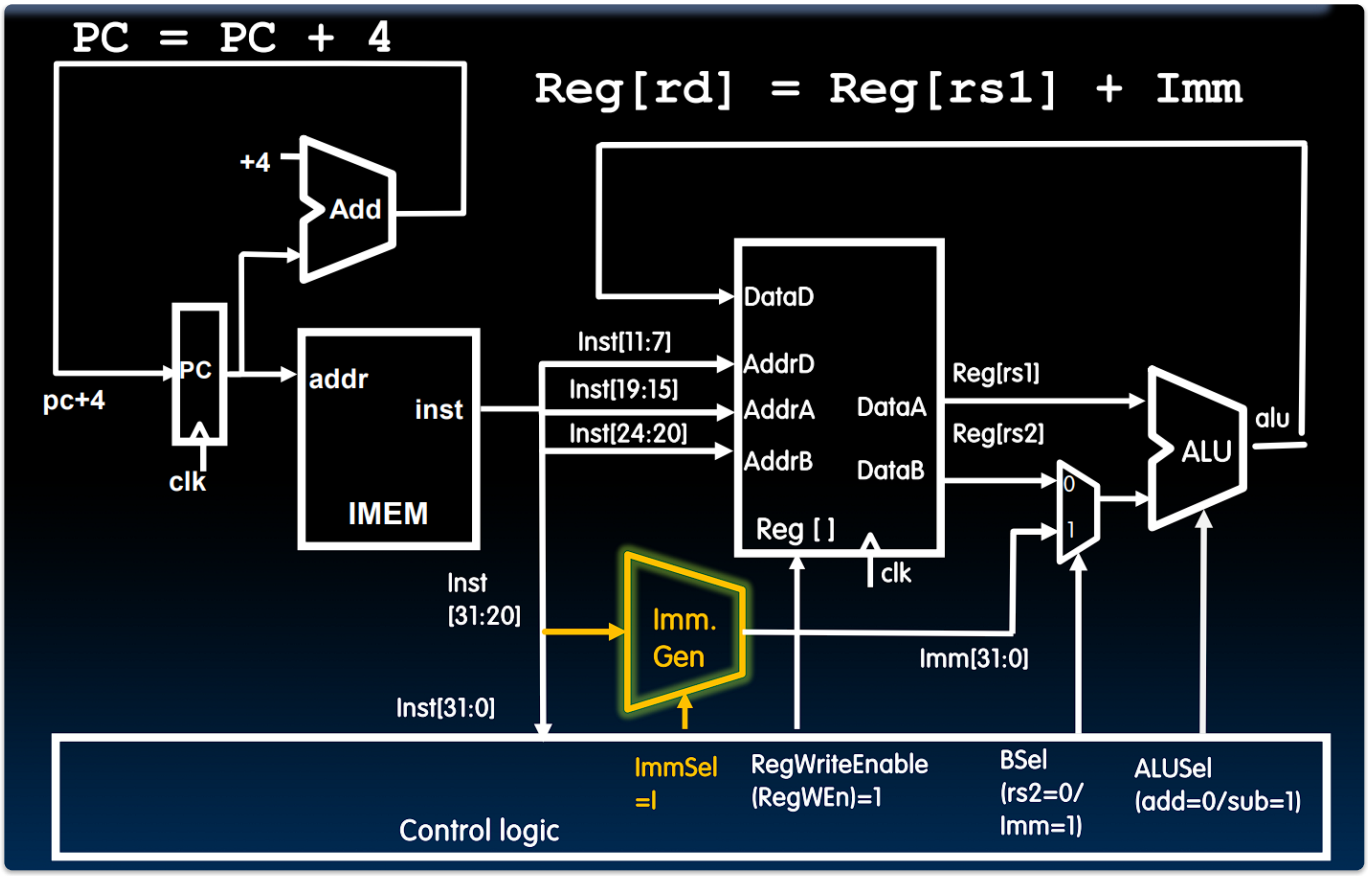
Datapath for add/sub

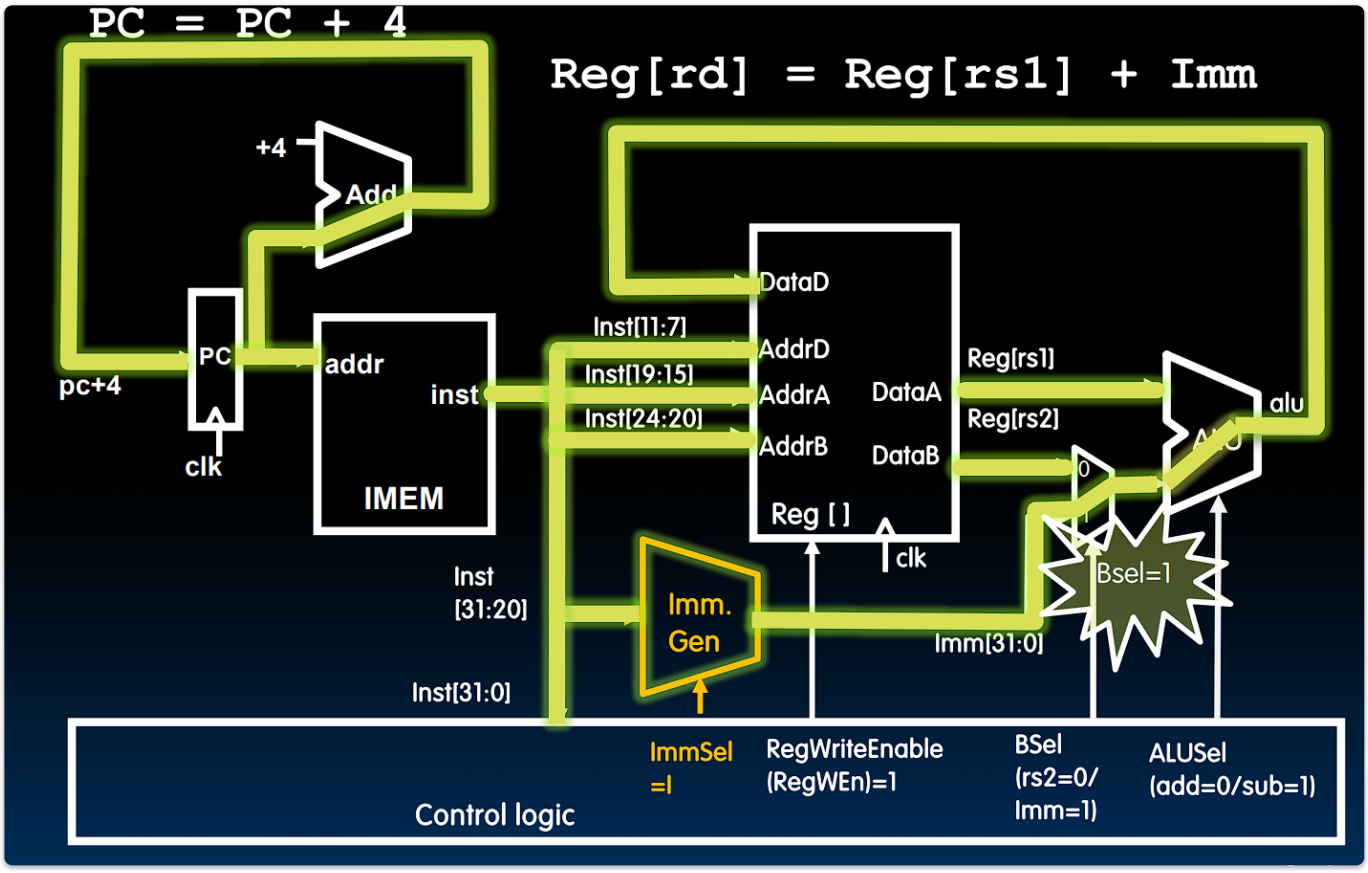


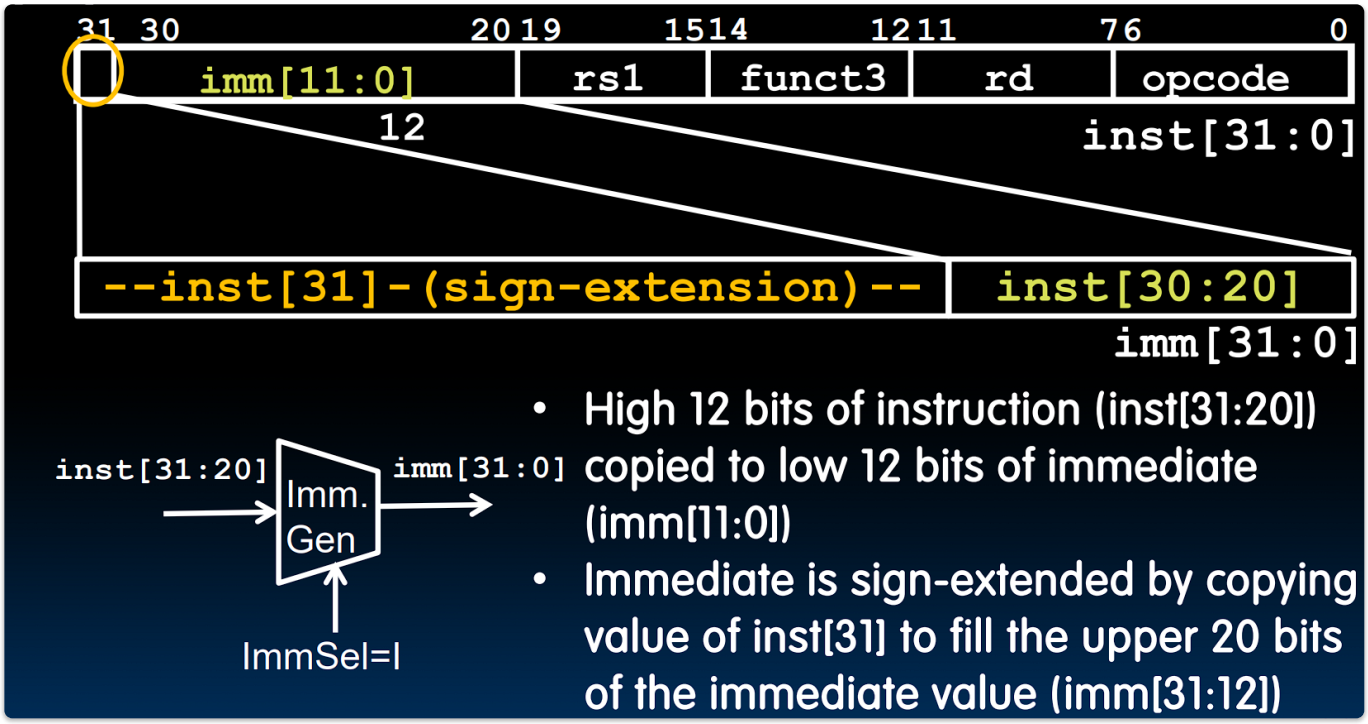
Add a Mux at there

Adding addi to Datapath




PC = PC + 4
Reg[rd] = Reg[rs1] + Imm
+4
Add
PC
pc+4
clk
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
0
1
ALU
alu
Inst
[31:20]
Imm.
Gen
Imm[31:0]
Inst[31:0]
ImmSel
=I
RegWriteEnable
(RegWEn)=1
BSel
(rs2=0/
Imm=1)
ALUSel
(add=0/sub=1)
Control logic


PC = PC + 4
Reg[rd] = Reg[rs1] + Imm
+4
Add
PC
pc+4
clk
addr
inst
IMEM
Inst[11:7]
Inst[19:15]
Inst[24:20]
DataD
AddrD
AddrA
AddrB
DataA
DataB
Reg [ ]
clk
Reg[rs1]
Reg[rs2]
0
alu
Bsel=1
Inst
[31:20]
Imm.
Gen
Imm[31:0]
Inst[31:0]
ImmSel
=I
RegWriteEnable
(RegWEn)=1
BSel
(rs2=0/
Imm=1)
ALUSel
(add=0/sub=1)
Control logic


31 30
20 19
15 14
12 11
7 6
0
imm[11:0]
rs1
funct3
rd
opcode
12
inst[31:0]
--inst[31]-(sign-extension)--
inst[30:20]
imm[31:0]
inst[31:20]
Imm. Gen
imm[31:0]
ImmSel=I
• High 12 bits of instruction (inst[31:20]) copied to low 12 bits of immediate (imm[11:0])
• Immediate is sign-extended by copying value of inst[31] to fill the upper 20 bits of the immediate value (imm[31:12])

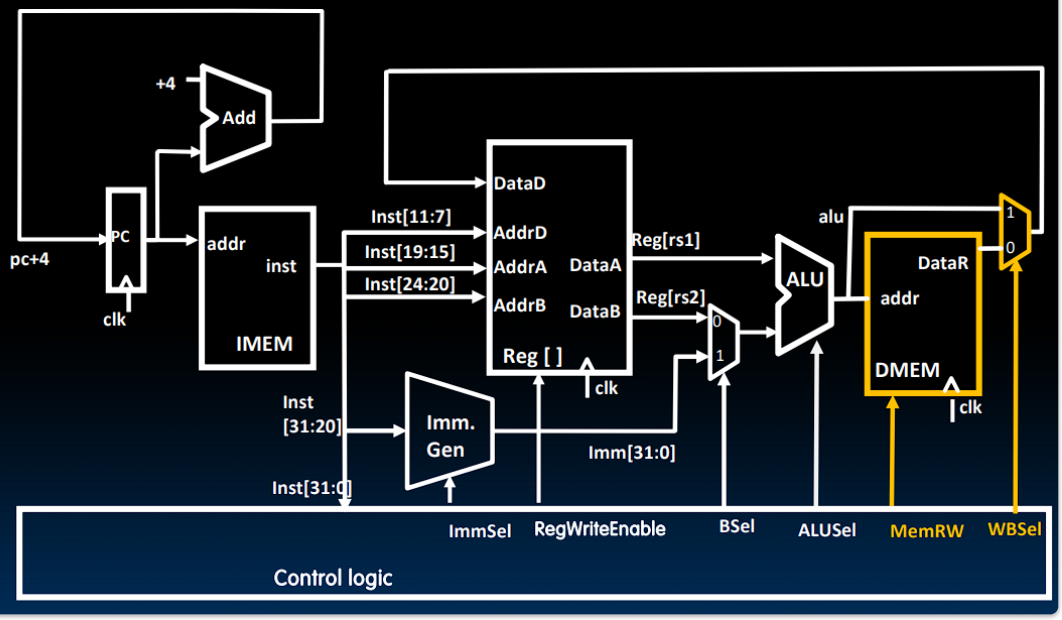
load

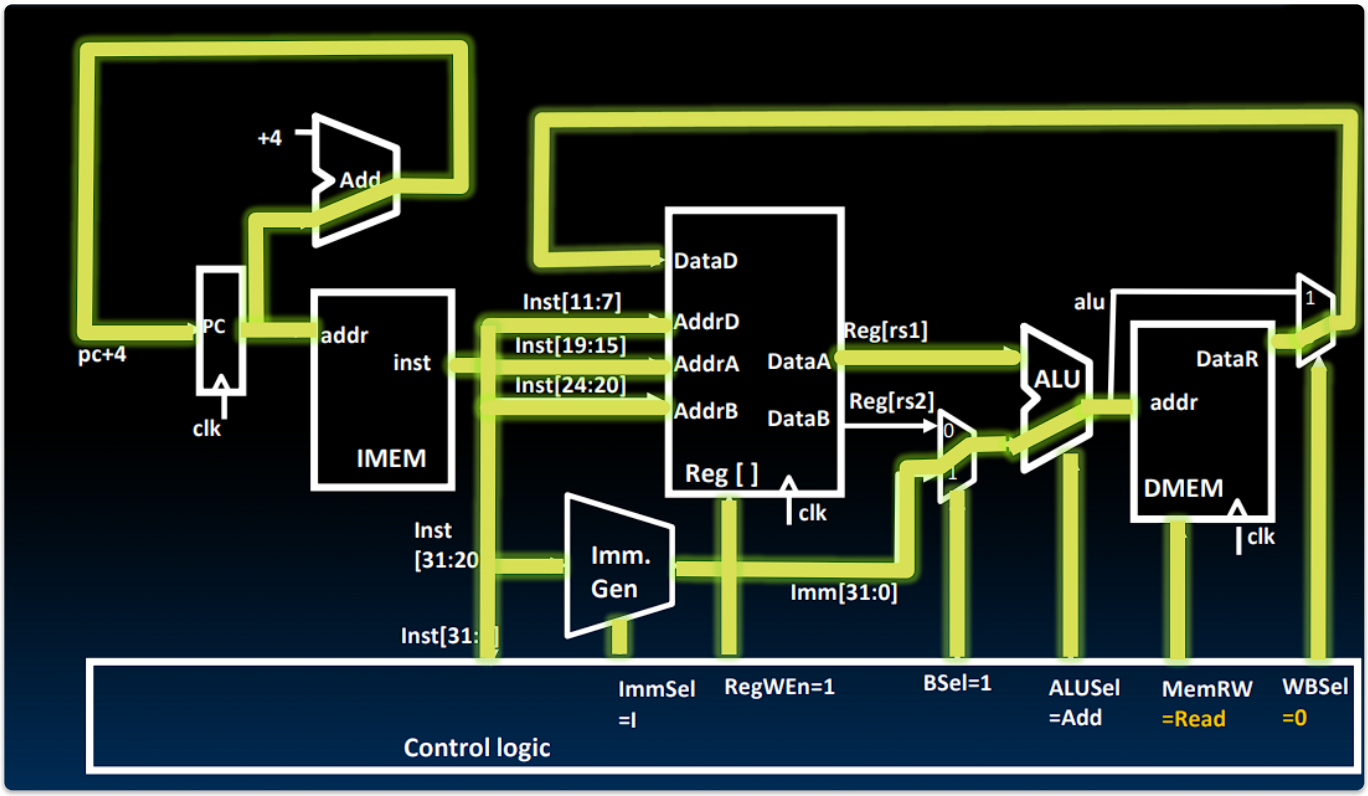
- RISC-V Assembly Instruction (I-type): `lw x14, 8(x2)`

31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD
000000001000	00010	010	01110	0000011
imm=+8	rs1=2	lw	rd=14	LOAD

- The 12-bit signed immediate is added to the base address in register **`rs1`** to form the memory address
  - This is very similar to the add-immediate operation but used to create address not to create final result

R+I Arithmetic/Logic Datapath





imm[11:0]	rs1	000	rd	0000011	lb
imm[11:0]	rs1	001	rd	0000011	lh
imm[11:0]	rs1	010	rd	0000011	lw
imm[11:0]	rs1	100	rd	0000011	lbu
imm[11:0]	rs1	101	rd	0000011	lhu

funct3 field encodes size and 'signedness' of load data

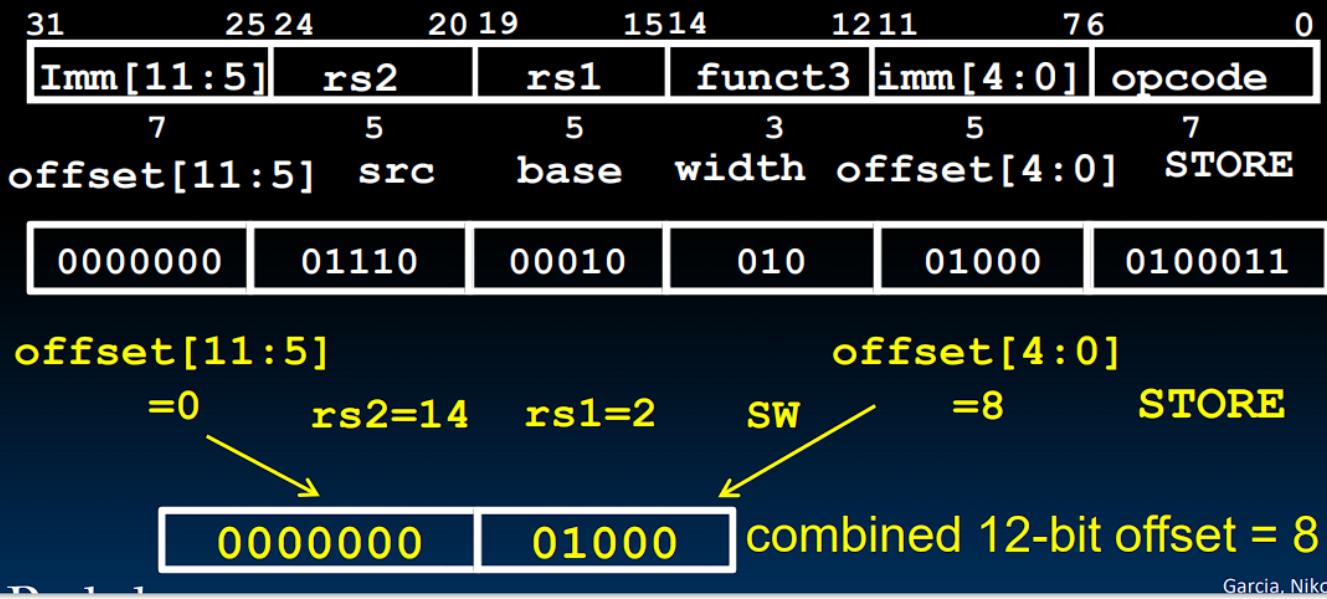
- Supporting the narrower loads requires additional logic to extract the correct byte/halfword from the value loaded from memory, and sign- or zero-extend the result to 32 bits before writing back to register file.
  - It is just a mux + a few gates

S-Format

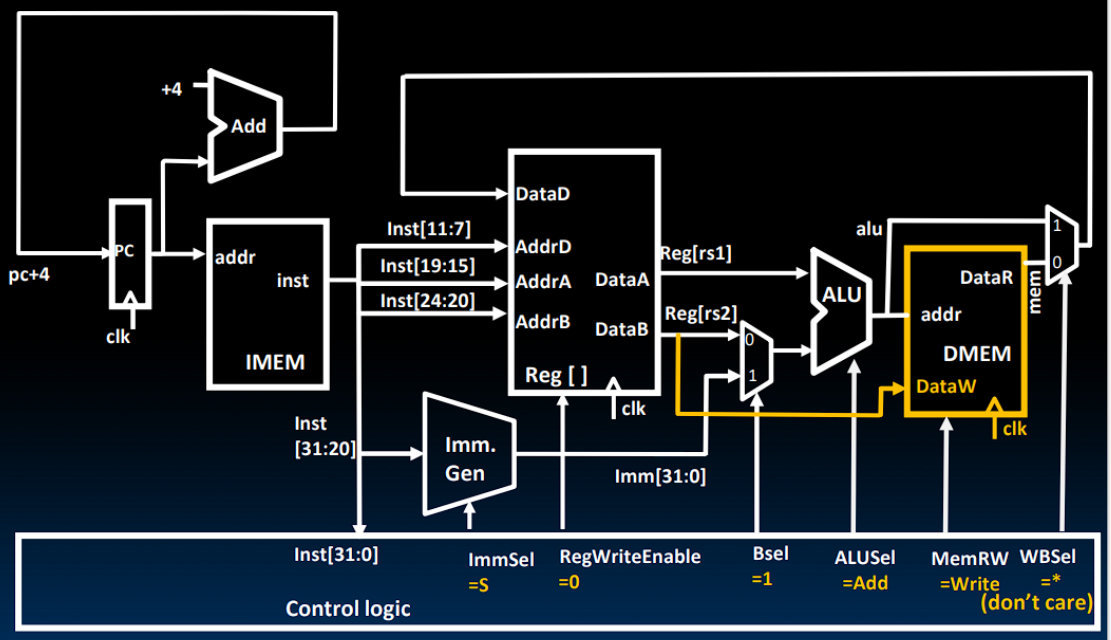
ADDING SW INSTRUCTION

- **sw**: Reads two registers, rs1 for base memory address, and rs2 for data to be stored, as well immediate offset!

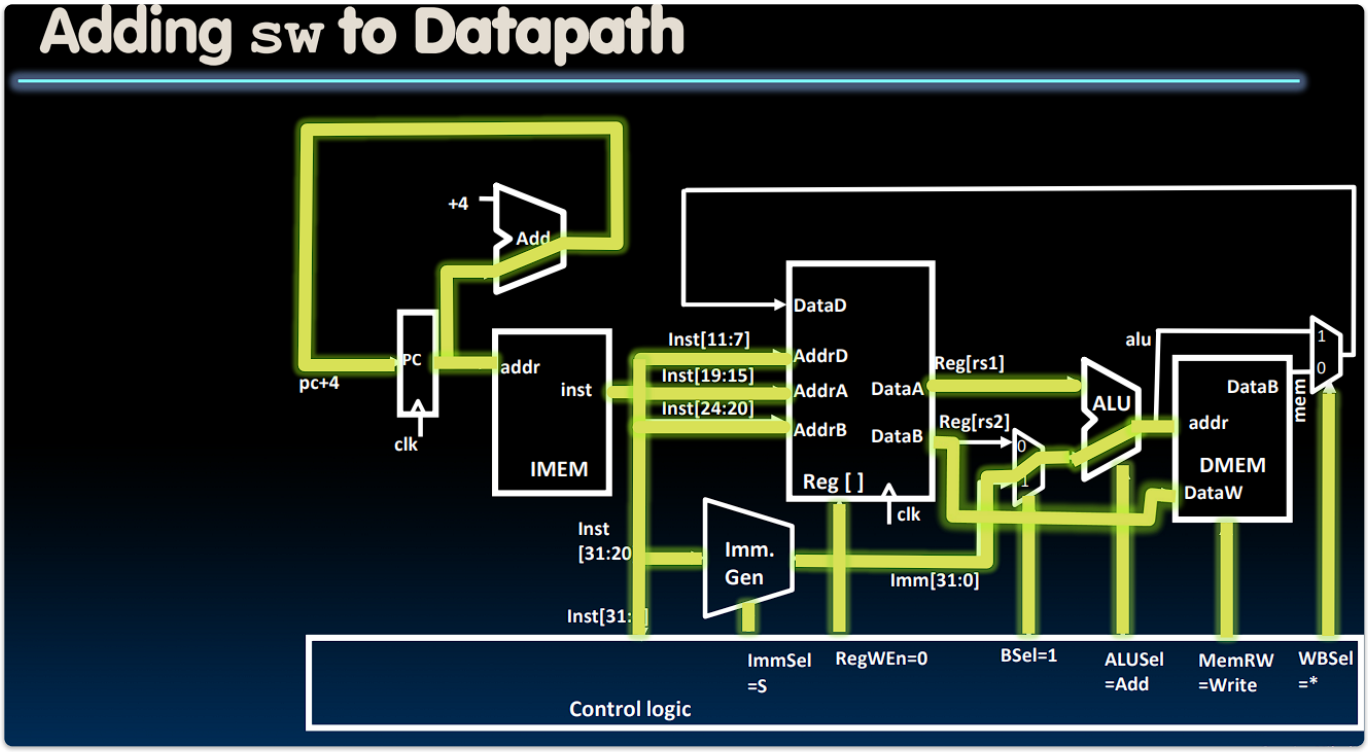
```
sw x14, 8(x2)
```



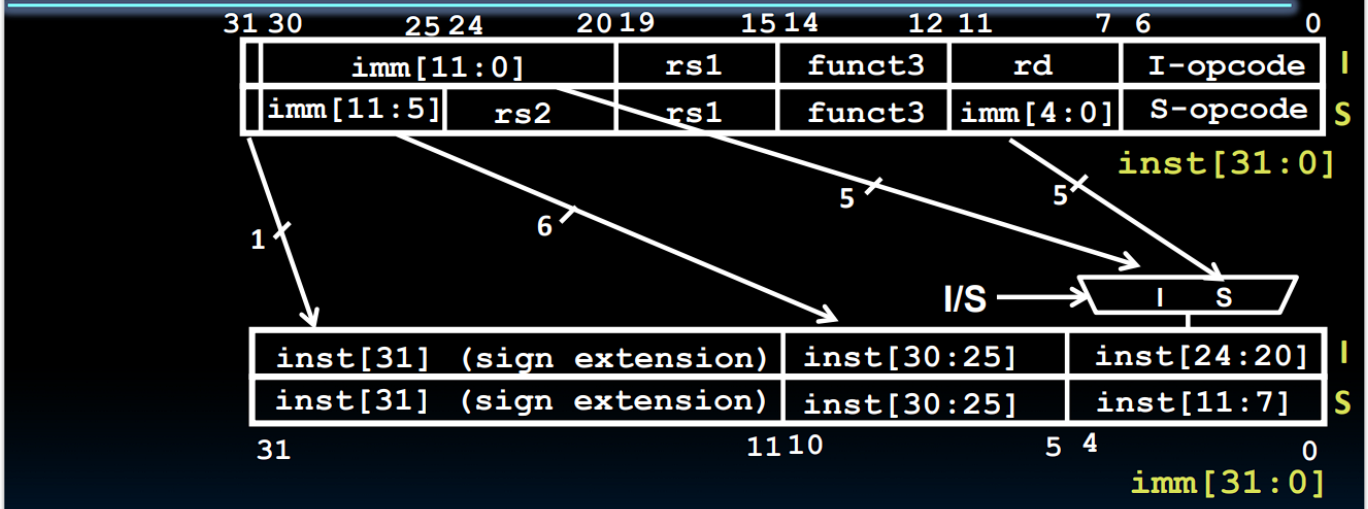
Adding sw to Datapath



Adding sw to Datapath

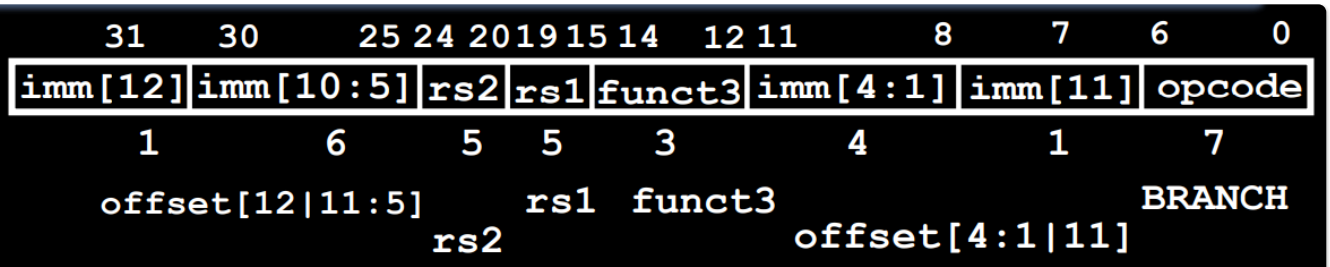


I+S Immediate Generation



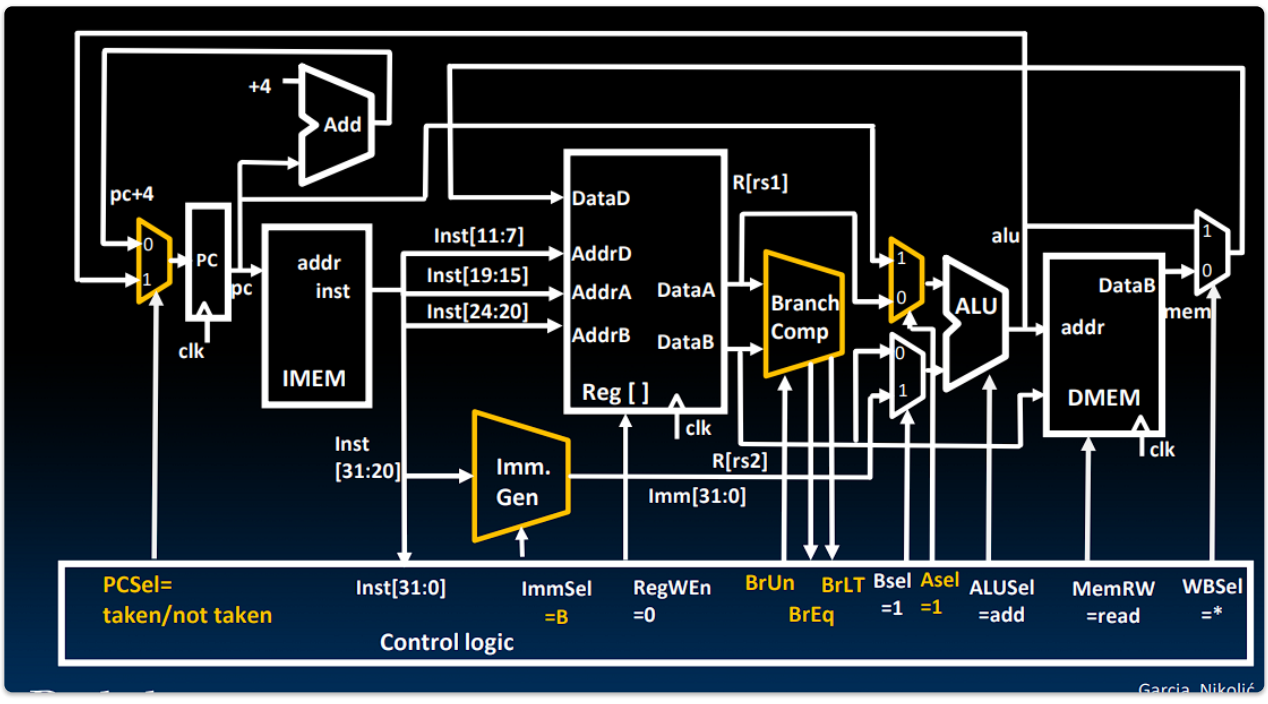
- Just need a 5-bit mux to select between two positions where low five bits of immediate can reside in instruction
- Other bits in immediate are wired to fixed positions in instruction

B-Format

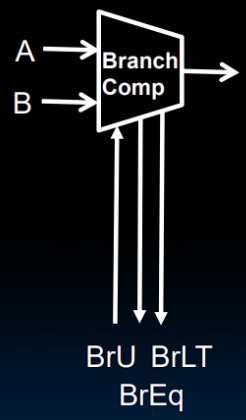


- B-format is mostly same as S-Format, with two register sources (**rs1**/**rs2**) and a 12-bit immediate **imm[12:1]**
- But now immediate represents values -4096 to +4094 in 2-byte increments
- The 12 immediate bits encode even 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it)

- Need to compute **PC+IMMEDICATE** and to compare values of **rs1 and rs2**



Branch Comparator



BrEq = 1, if A=B

BrLT = 1, if A<B

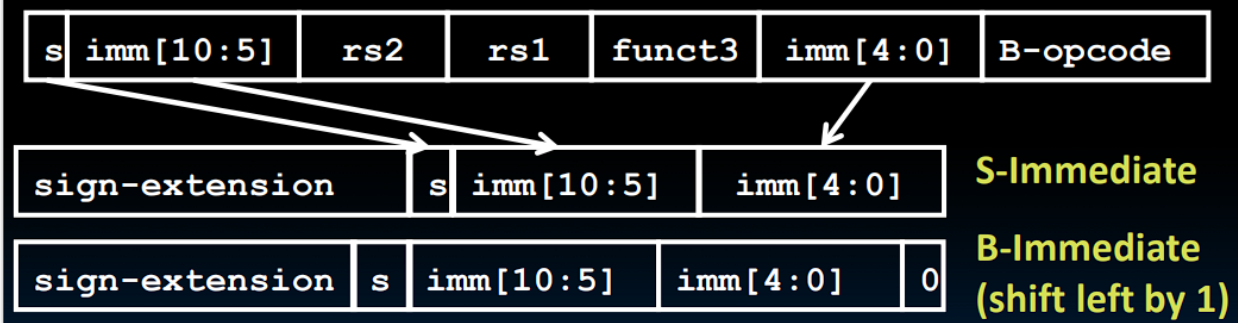
BrUn = 1 selects unsigned comparison for **BrLT**, 0=signed

BGE branch: A >= B, if $\overline{A<B}$

$\overline{A<B}$ = !(A<B)

12-bit immediate encodes PC-relative offset of -4096 to +4094 bytes in multiples of 2 bytes

Standard approach: Treat immediate as in range -2048..+2047, then shift left by 1 bit to multiply by 2 for branches



Each instruction immediate bit can appear in one of two places in output immediate value – so need one 2-way mux per bit

Lighting Up Branch Path