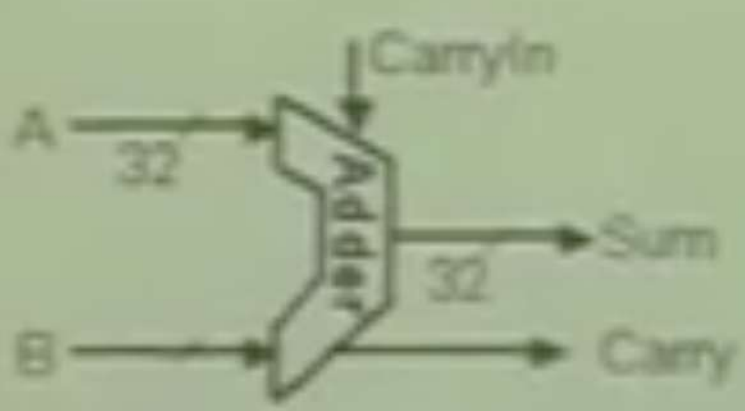
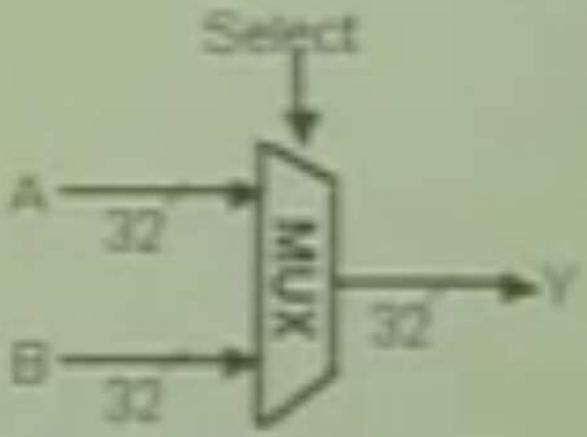
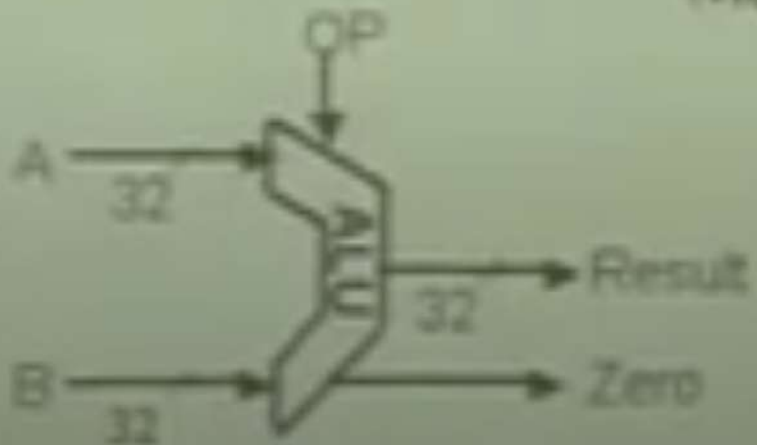
Combinational Elements



Adder



MUX

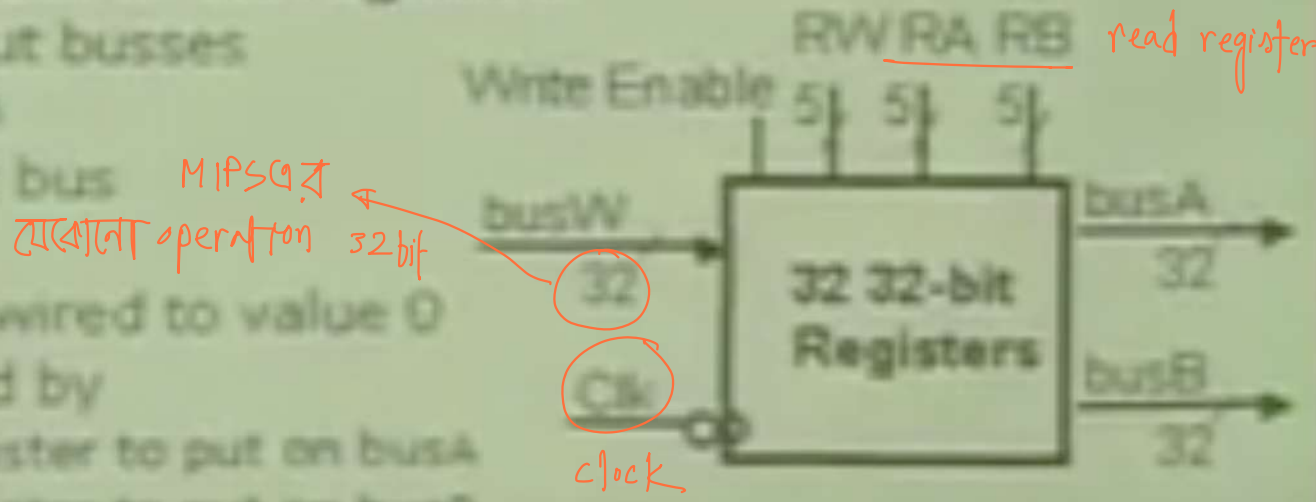


ALU

Storage Element: Reg File

RW → write register

- Register File consists of 32 registers
  - Two 32 bit output busses
    - busA and busB
  - One 32 bit input bus
    - busW
  - Register 0 hard wired to value 0
  - Register selected by
    - RA selects register to put on busA
    - RB selects register to put on busB
    - RW selects register to be written via busW when Write Enable is 1
  - Clock input (CLK)
    - CLK input is a factor only for write operation
    - During read, behaves as combinational logic block
      - RA or RB stable ⇒ busA or busB valid after "access time"
      - Minor simplification of reality



read is a combinatorial operation. → i/p দিলে o/p পাওয়া যায়

- clock লাগে না।
- delay হয় না।

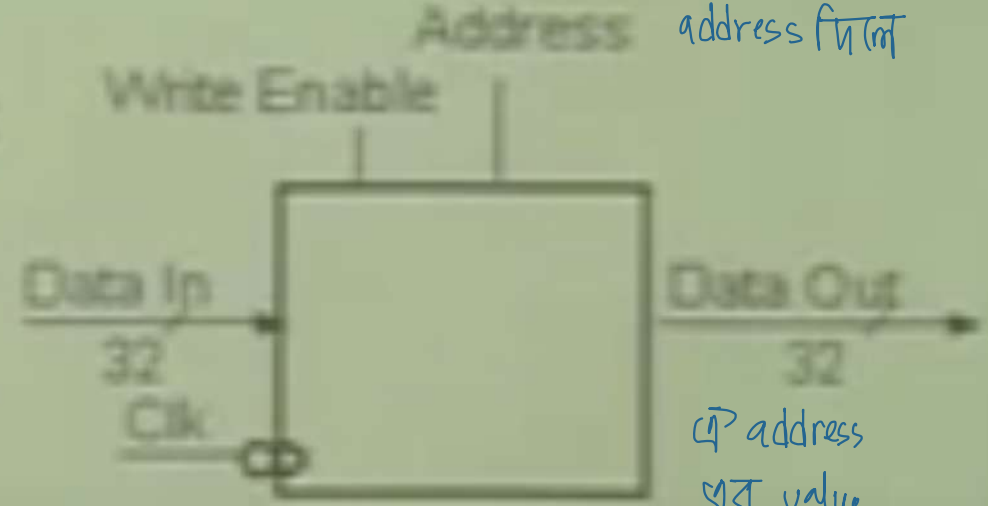
write is a sequential operation.

↳ trigger করা হলে operation perform করে।

Storage Element: Memory

memory র abstraction

- Memory
  - One input bus: Data In
  - One output bus: Data Out
  - Address selection
    - Address selects the word to put on Data Out
    - To write to address, set Write Enable to 1
  - Clock input (CLK)
    - CLK input is a factor only for write operation
    - During read, behaves as combinational logic block
      - Valid Address → Data Out valid after "access time"
      - Minor simplification of reality



address দিলে
address এর value পাবো।
(read এ)

Write enable এ 1

Data in দিবো

triggered হলে address এ সেই value write হবে।

$2^{32} \times 4$ byte জায়গা আছে

→ 2^{32} বা 4G সংখ্যক address/location আছে। প্রতিটা location এ 4

আগে একটা single instruction এ write, read করা যেত। → R format (reg file এ)

but memory তে একই সাথে দুটো কাজ করা যায় না। datapath একটা

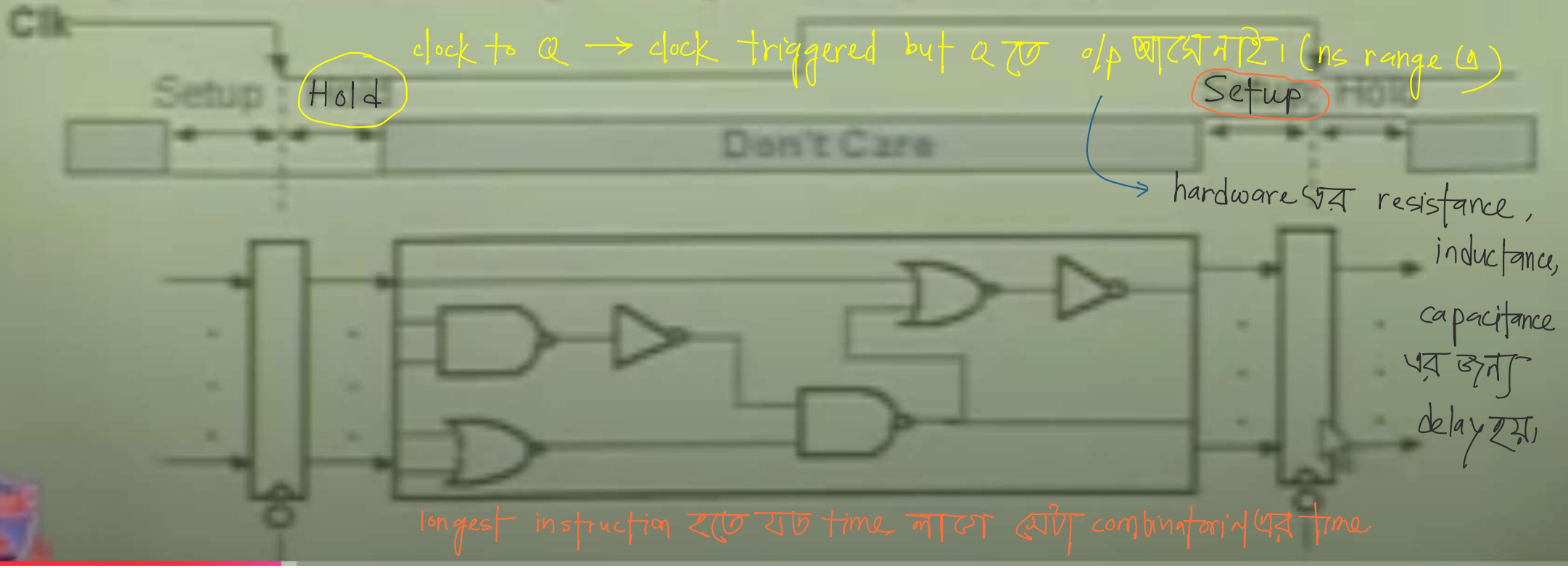
একটা wire দিয়ে অনেকগুলো data একই সাথে যেতে পারে।

multiplexing

Some Logic Design...

- **All storage elements have same clock**
  - Edge-triggered clocking
  - "Instantaneous" state change (simplification!)
  - Timing always work if the clock is slow enough

Cycle Time = Clk-to-Q + Longest Delay + Setup + Clock Skew



combinatorial এর কাজ → clock এর significant time আগে Flip Flop এর data ready রাখতে হবে। → set up

যেকোনো processor এর
- fetch — instruction fetch
- decode — 6 bit opcode, 5 bit register, ...
- execute

decode → machine কোড ভেঙে
① বুঝতে হবে কোনটা register, কোনটা কি,

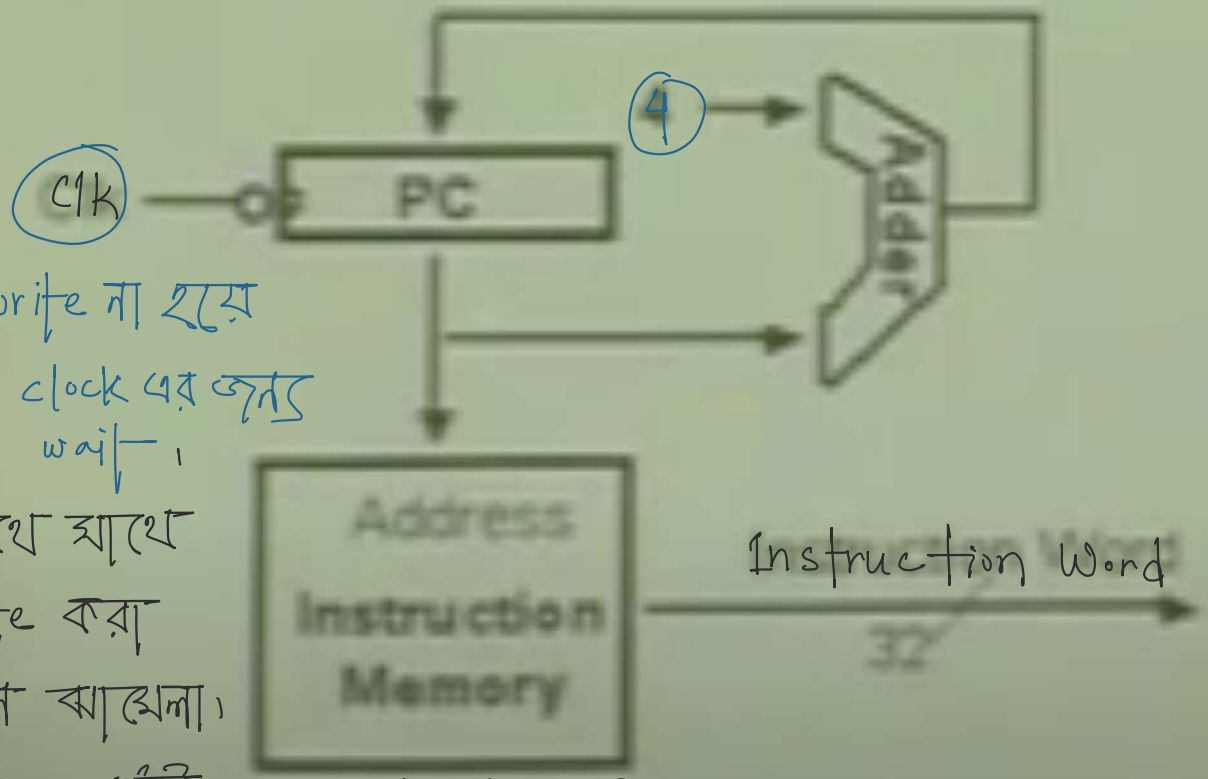
instruction আনার পর
যা যা করবে হবে execution.

fetching : PC register এ যে address থাকে সেটা নিয়ে instruction memory তে যাবে।

flat memory logically data and code
minimum এই ২ ভাগে ভাগ করে।

code — instruction memory

Datapath: IF Unit



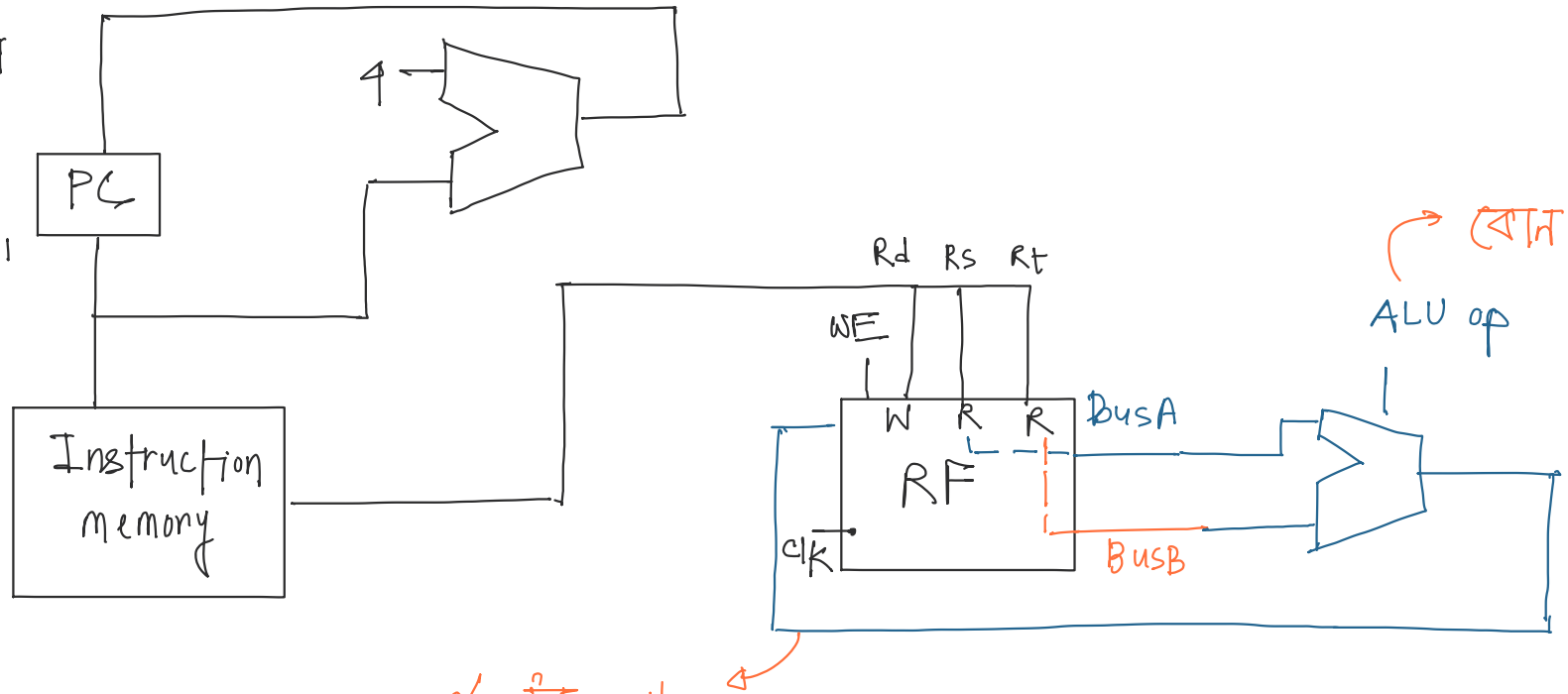
write না হয়ে clock এর জন্য wait।

যাথে যাথে write করা গেলে ঝামেলা।

because এটাই next instruction কিনা সেটা guaranteed না। jumping আছে, branching আছে।

এখানেই fetch কোস না পারের টার জন্য ready থাকতে হবে।

PC+4



PC+4 আসবে। যাতে যাতে write হবে না। clock pulse পাবে। branch, jump আছে। So PC+4 is not the guarrented next register

ALU কোন operation হবে।

o/p টা write এর জন্য যাবে। WE (write enable pin) 1 থাকলে Rd এ o/p টা write হবে।

R type instruction দিয়ে implement করছি।

add $t0 , $t1 , $t2 — R format

Rd Rs Rt

$t0 = $t1 + $t2

BusA দিয়ে যাবে Rs এর value
BusB " " Rt " " } ALU তে যাবে।

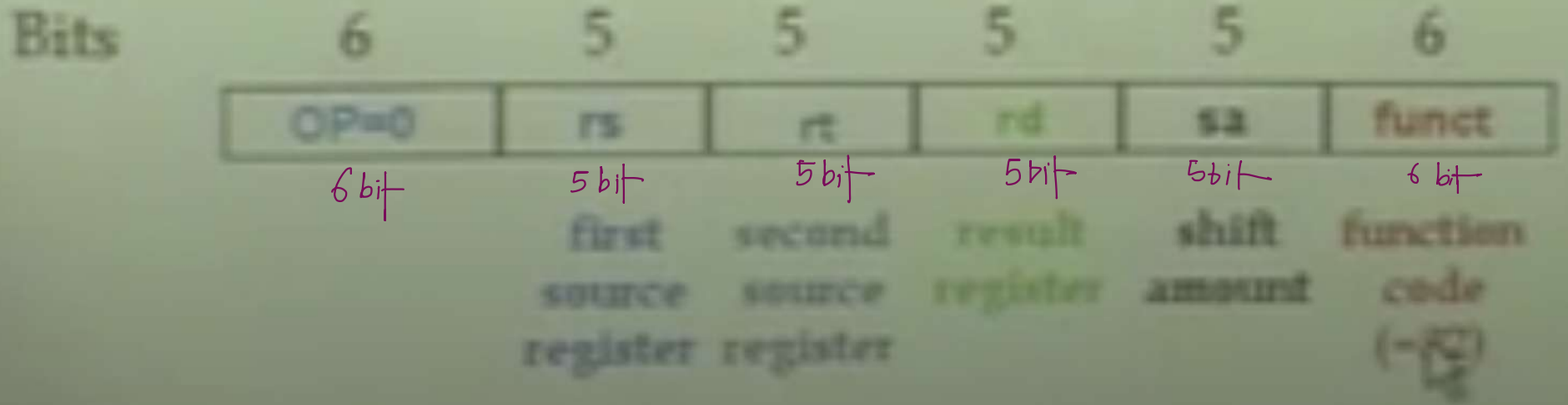
Add RTL

addition — arithmatic

- Add instruction

```
add rd, rs, rt
```

```
Mem[PC]:          Fetch instruction from memory
R[rd] <- R[rs] + R[rt]:   Add operation
PC <- PC + 4;     Calculate next address
```

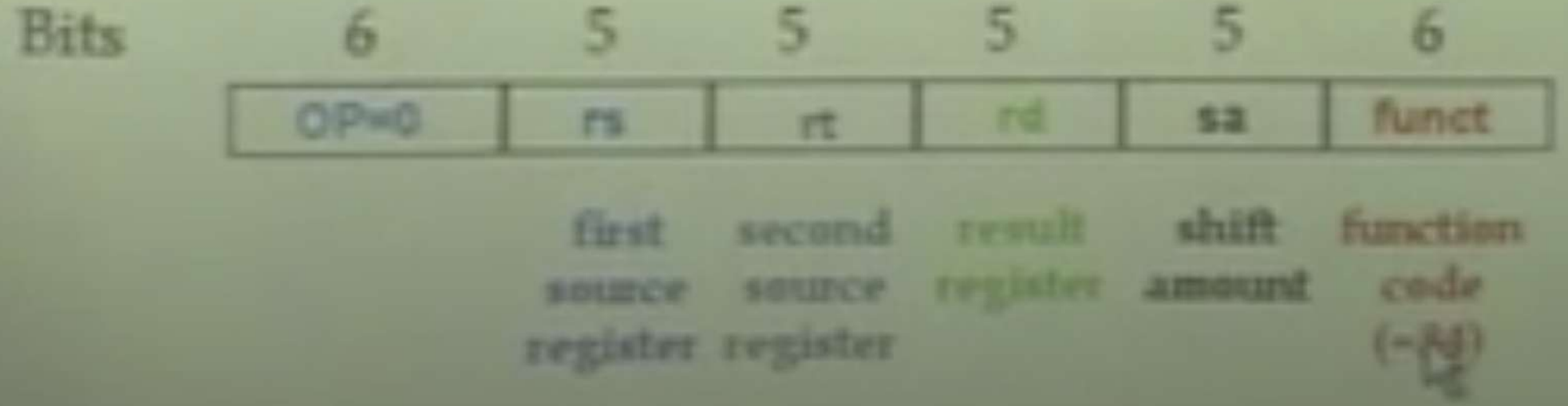


Sub RTL

- Sub instruction

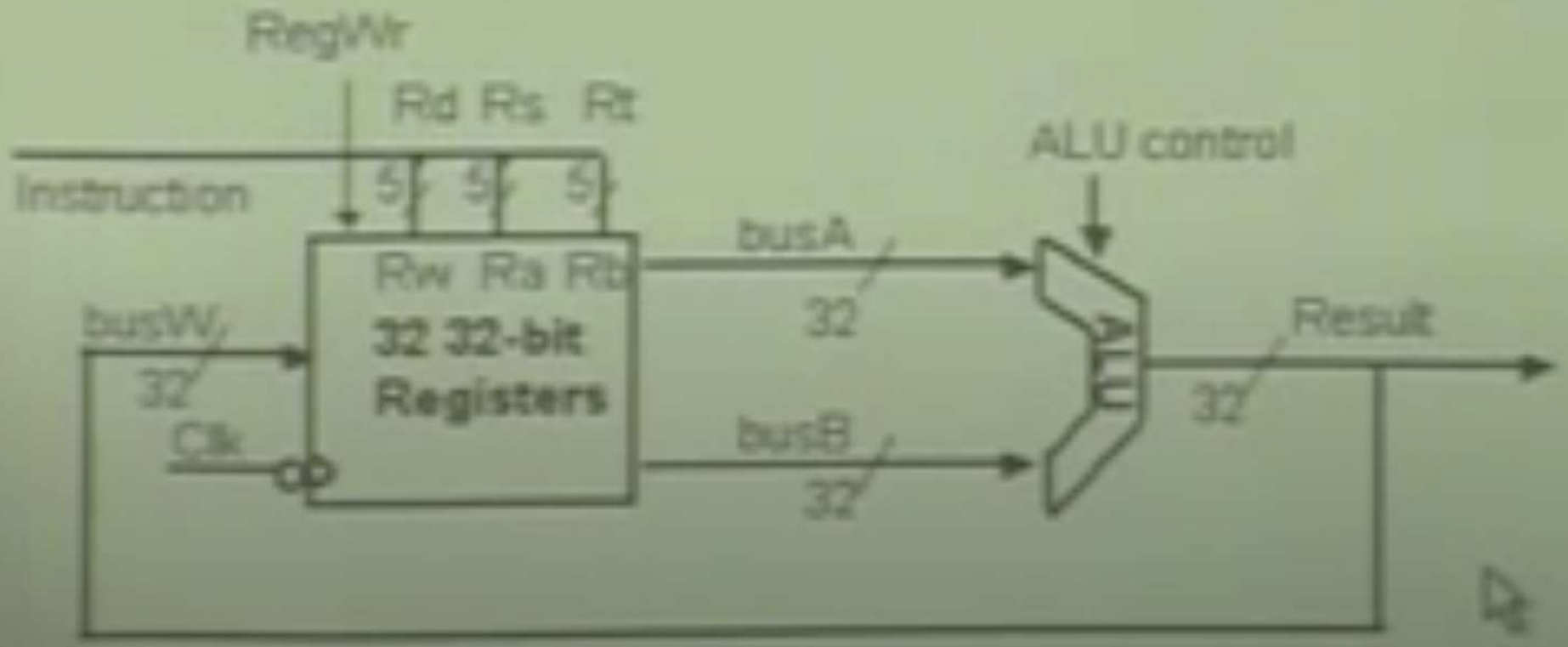
```
sub rd, rs, rt
```

```
Mem[PC]:                    Fetch instruction from memory
R[rd] <- R[rs] - R[rt]:     Sub operation
PC <- PC + 4:               Calculate next address
```



Datapath: Reg/Reg Ops

- `R[rd] <- R[rs] op R[rt];`
  - ALU control and RegWr based on decoded instruction
  - Ra, Rb, and Rd from rs, rt, rd fields



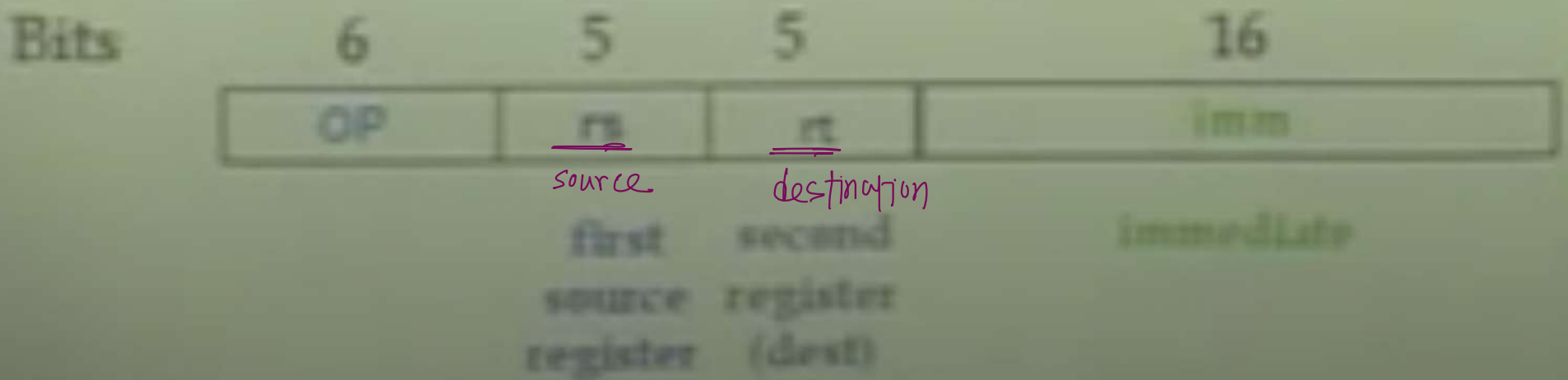
OR Immediate RTL

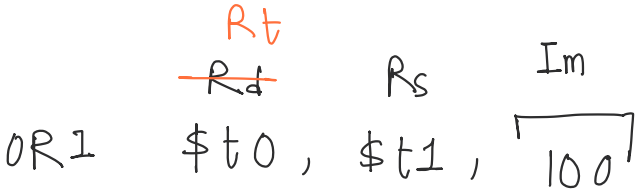
or immediate
— I format

- OR Immediate instruction

```
ori rt, rs, imm

Mem[PC];                          Fetch instruction from memory
R[rt] <- R[rs] OR ZeroExt(imm);
                                  OR operation with Zero-Extend
PC <- PC + 4;                     Calculate next address
```



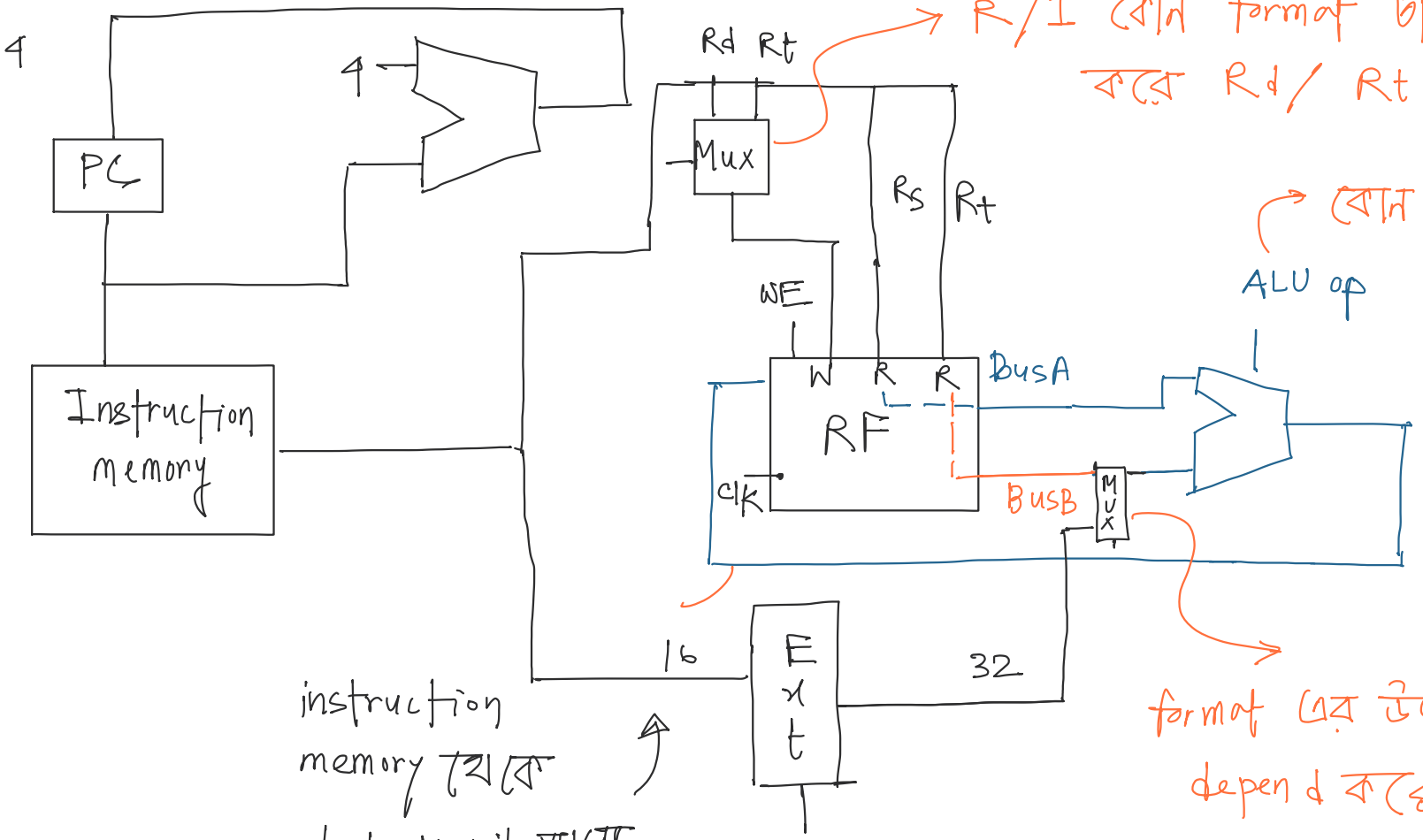


I format এ কোনো Rd নাই
Rt টাই destination হবে।

ori এ write operation আছে।

↳ এর peripheral এ রাখি। Rd না, Rt।

কিভাবে করবো? → mux দিয়ে



R/I কোন format তার উপর depend করে Rd/ Rt যাবে।

কোন operation হবে।

instruction memory থেকে last 16 bit পাবো যেটা ALU তে যাবে। immediate এর।

format এর উপর depend করে।

R, I — Ok

arithmatic operation ⟶ sign bit দিয়ে extension

আমনের 16 bit (32bit ALU)

↓

immediate 16 bit বাকিটা এভাবে extension.

logical " ⟶ 0 বসাবো

0 extension

Load

lw $t0, 80($s0)

Rt Rs

Rt = MEM[80 + Rs]

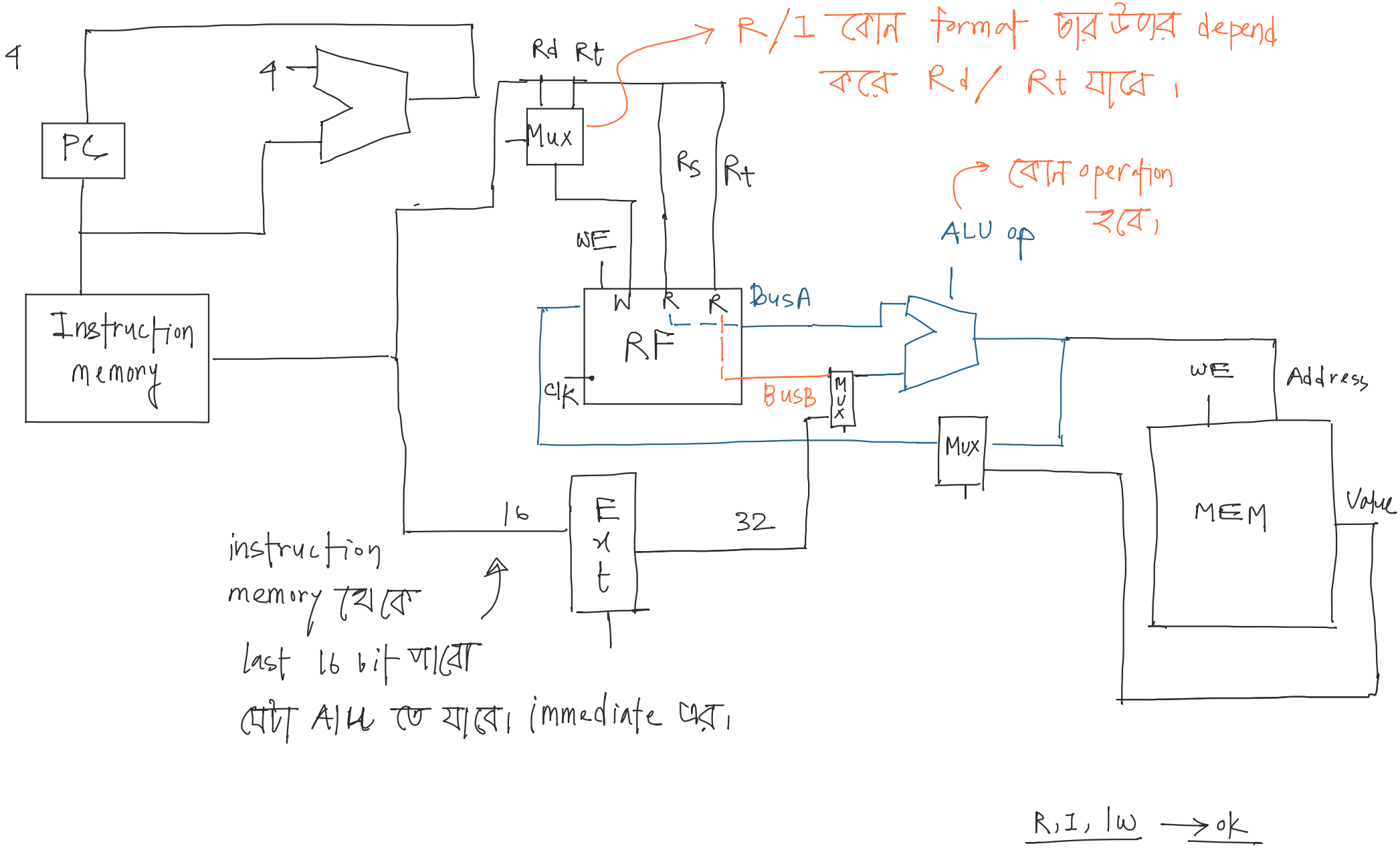
↳ alu র o/p direct write না আগে memory তে যাবে।

write operation আছে। Rt তে write হয়।

ঠিকঠাক select হয়। ok

add Rs and immediate ⟹ এগুলোও ঠিকঠাক যাবে।

but address টা আমরা write করবো না।


4
4
PC
Instruction memory
Rd Rt
Mux
Rs
Rt
WE
W
R
R
RF
clk
BusA
BusB
MUX
ALU op
Mux
WE
Address
MEM
Value
16
Ext
32
R/I কোন format তার উপর depend করে Rd/ Rt যাবে।
কোন operation হবে।
instruction memory থেকে last 16 bit গুলো যেটা ALU তে যাবে। immediate এর।
R, I, lw → ok

80 + Rs ⟶ memory address

memory তে যাবে। যাথে যাথে read হবে। কারন read combinatorial operation.
সেই value টা Rt তে write হবে।

Store: sw $t0, 80($s0)

Rt Rs

Rt = MEM[80 + Rs]

register write নাই।

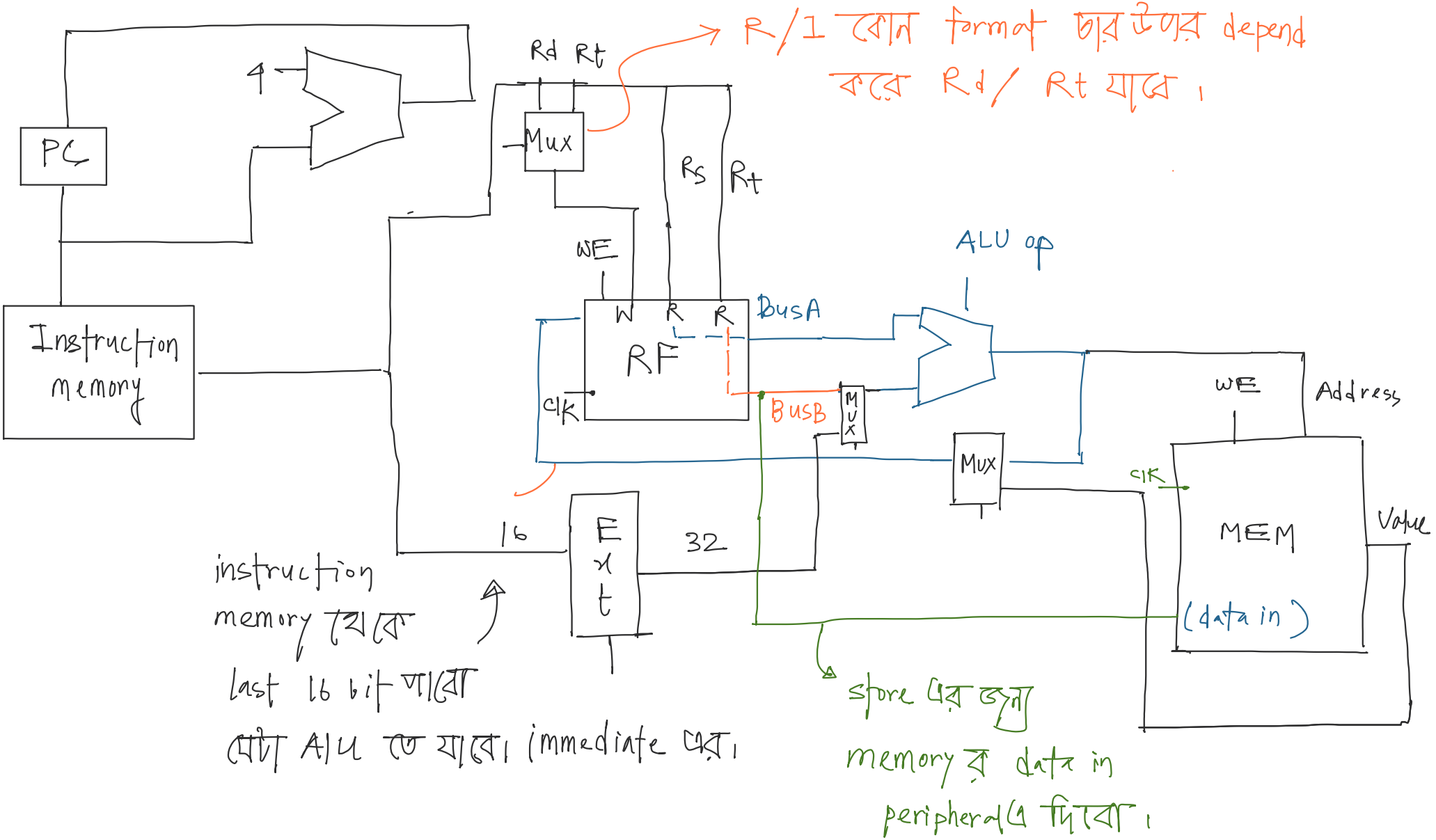
W তে কোনো কাজেলা নেই :(RF এর)

memory র write enable pin 1 হবে।

এখন যা write করবো সেটা লাগবে। BusB থেকে পাবো।

Rt write করবো

R, I, sw, lw → ok


R/I কোন format তার উপর depend করে Rd/ Rt যাবে।
Rd Rt
Mux
PC
Rs
Rt
WE
ALU op
Instruction memory
W R R
RF
clk
BusA
BusB
MUX
Mux
WE
Address
clk
MEM
Value
(data in)
16
Ext
32
instruction memory থেকে last 16 bit পাবো যেটা ALU তে যাবে। immediate এর।
store এর জন্য memory র data in peripheral এ দিবো।

Branch :

beq $t1, $t2, 100

Rt Rs → matter করে না but ori এর format ঠিক রাখতে চাই।

instruction টা standardize করলাম

register file এ no write

- ALU তে sub হবে। Rs, Rt এর।
  - ↳ compare এর জন্য।

- Rs, Rt ঠিক ভাবে যাবে।

ALU র o/p দেখে কাজ নেই। 2 bit দেখা লাগবে। plus এটা branch operation কি না দেখা লাগবে।

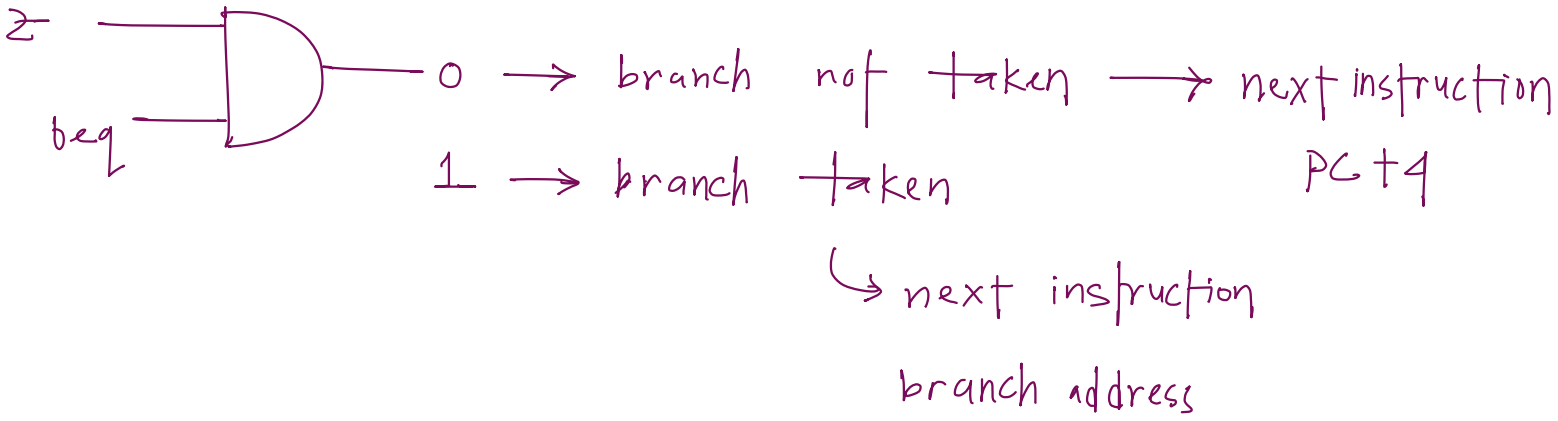
pc+4 + branch immediate add হয়ে mux এ যাবে। → branch address হবে।

pc+4 ও same mux এ যাবে।

beq (and operation) mux এর selection।

PC+4 mux এর 0 তে।

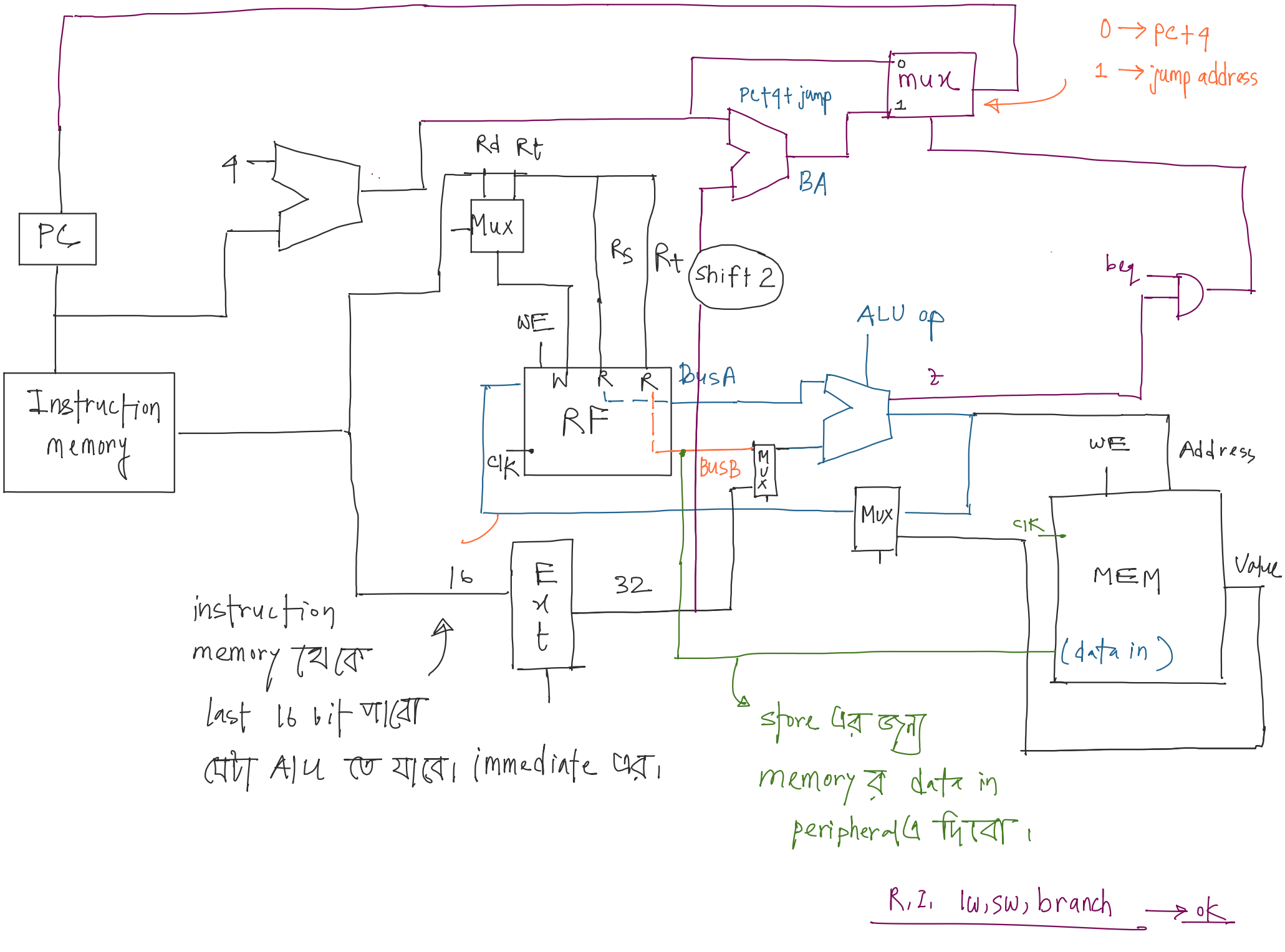
but immediate এর ২ ঘর shift করা লাগবে। left shift ২ ঘর।



next instruction address pc তে যাবে।

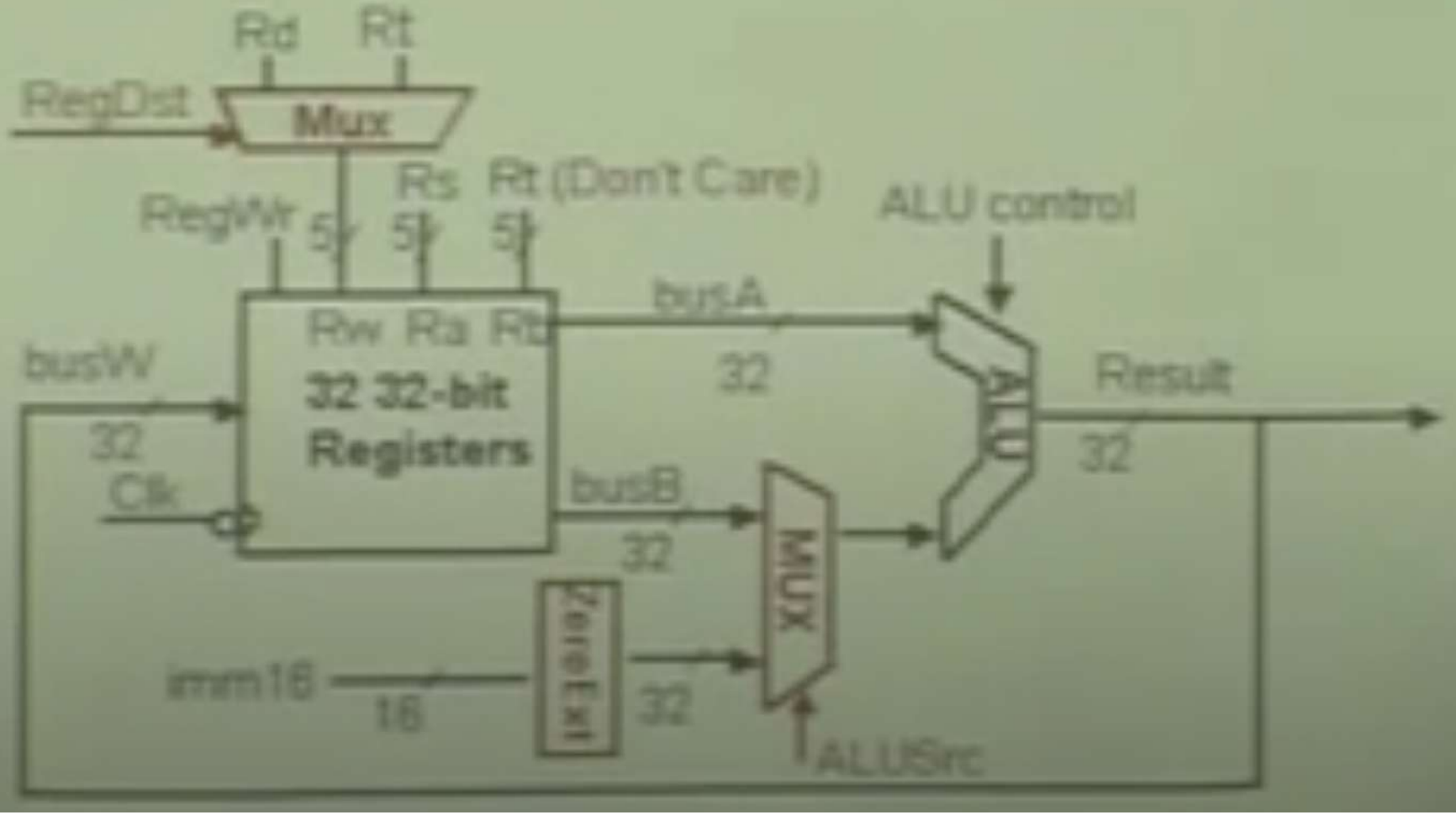
but write হবে না। write করতে clock লাগবে।

যদি clock নাও থাকতো তাও write করতে পারবো না। কারন branch address এখনো calculate হয় নাই। unconditional jump এখনো বাকি।


0 → pc+4
1 → jump address
mux
pc+4+ jump
BA
4
Rd Rt
Mux
PC
Rs
Rt
shift 2
beq
WE
ALU op
Instruction memory
W R R
RF
BusA
z
clk
BusB
MUX
WE
Address
Mux
clk
MEM
Value
(data in)
16
Ext
32
instruction memory থেকে last 16 bit পাবো যেটা ALU তে যাবে। immediate এর।
store এর জন্য memory র data in peripheral এ দিবো।
R, I, lw, sw, branch → ok

Datapath: Immediate Ops

- Rw set by MUX and ALU B set as busB or ZeroExt(imm)
- ALUsrc and RegDst set based on instruction

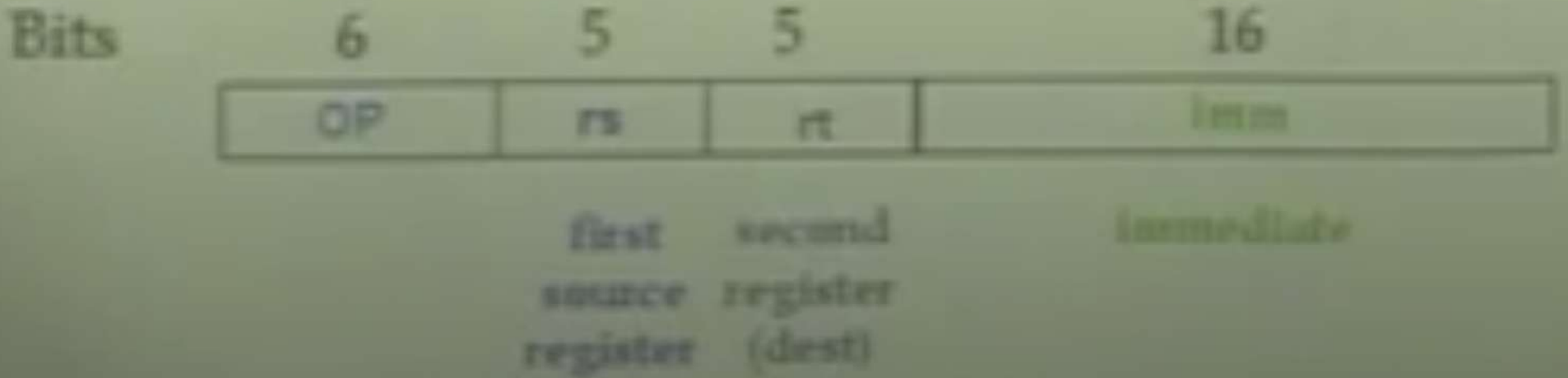


Load RTL

- Load instruction

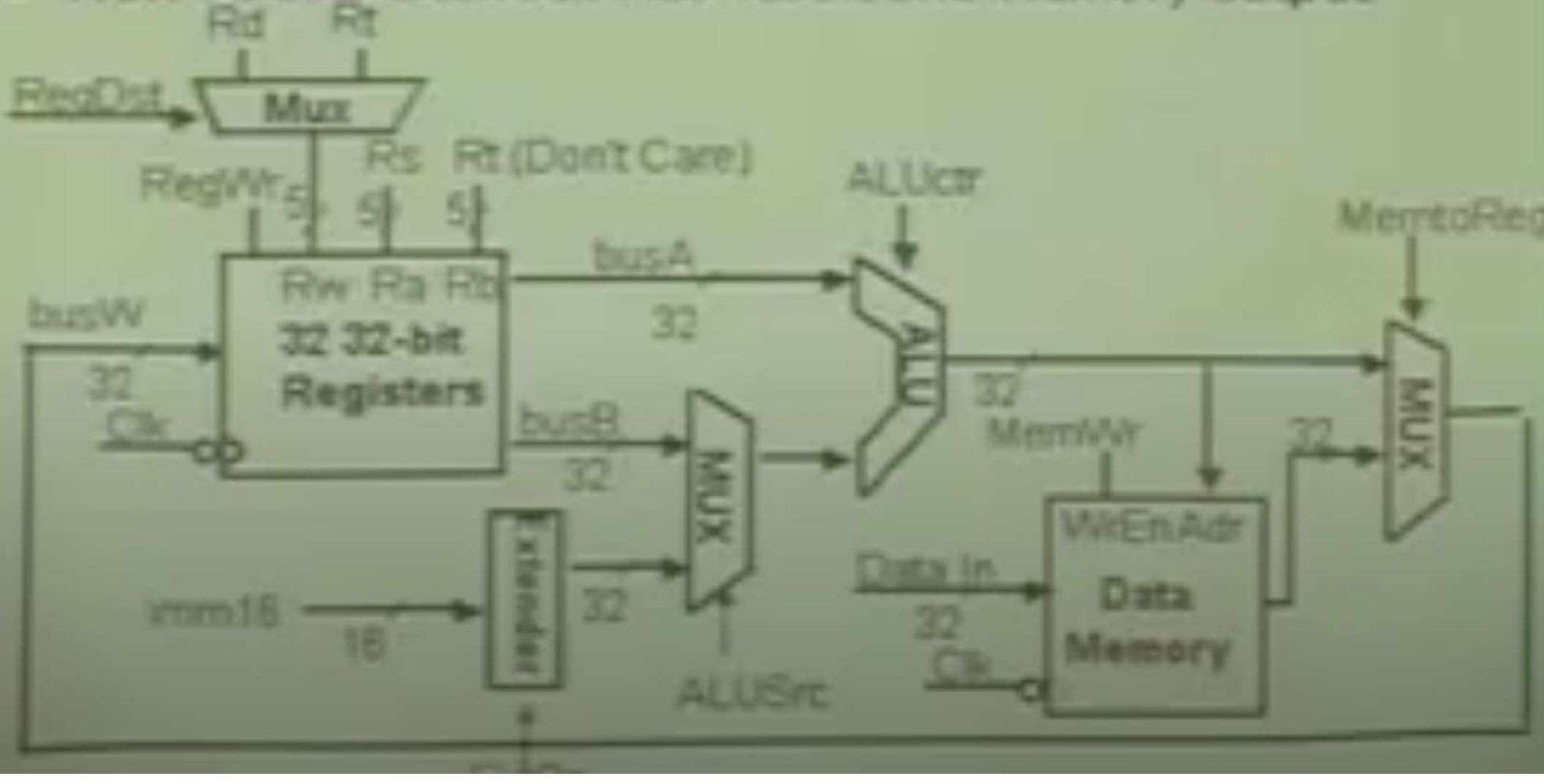
```
lw rt, rs, imm
```

```
Mem[PC];                          Fetch instruction from memory
Addr <- R[rs]+SignExt(imm);       Compute memory addr
R[rt] <- Mem[Addr];               Load data into register
PC <- PC + 4;                     Calculate next address
```



Datapath: Load

- Extender handles sign vs. zero extension of immediate
- MUX selects between ALU result and Memory output

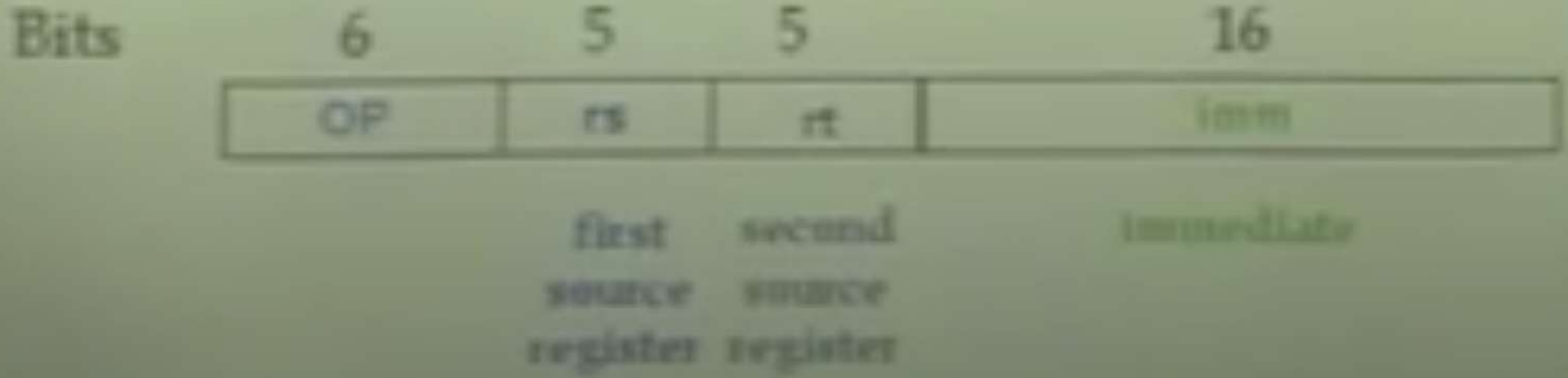


Store RTL

- Store instruction

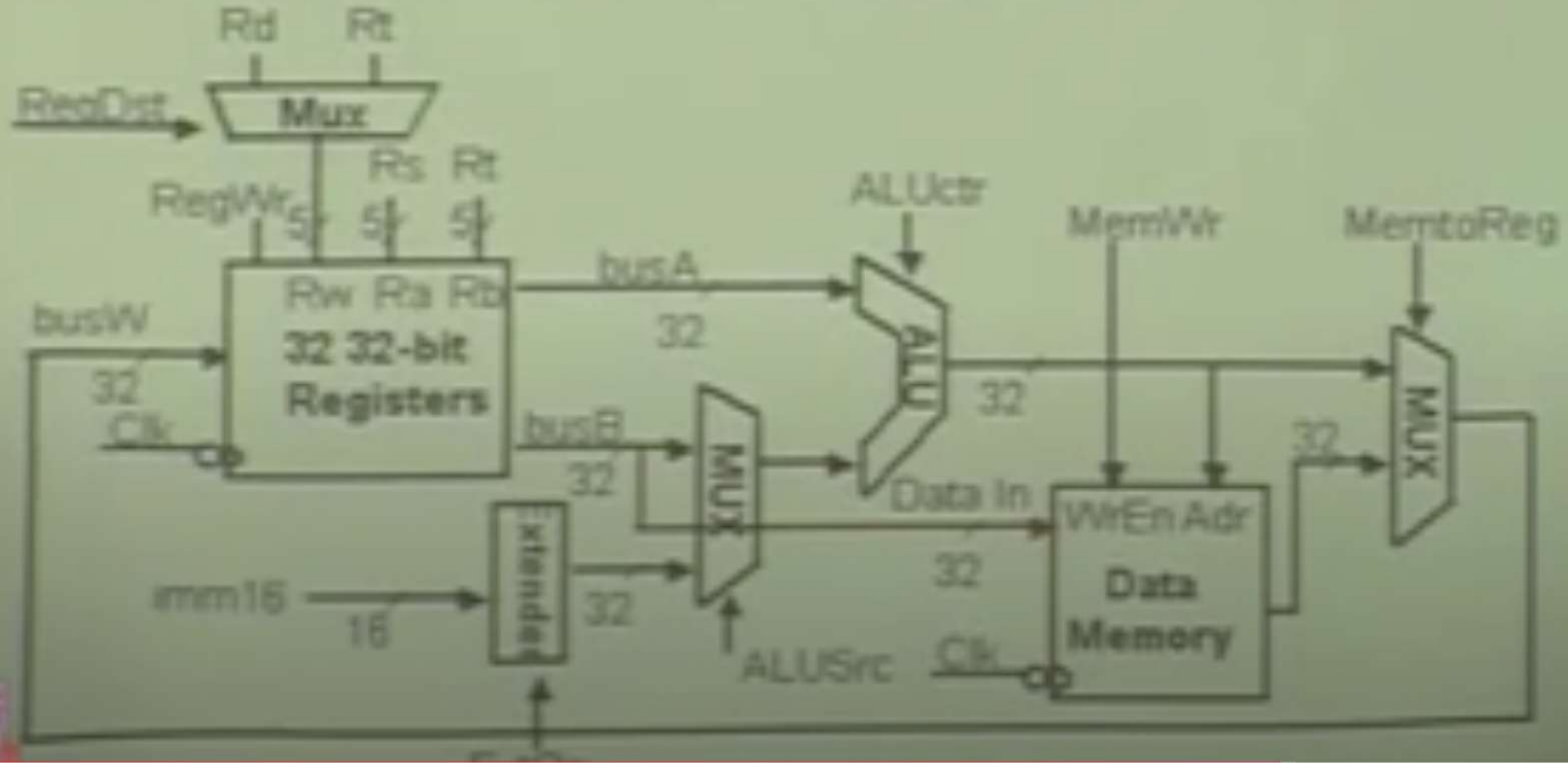
```
sw rt, rs, imm
```

```
Mem[PC]:                          Fetch instruction from memory
Addr <- R[rs]+ SignExt(imm):      Compute memory addr
Mem[Addr] <- R[rt]:               Load data into register
PC <- PC + 4:                     Calculate next address
```



Datapath: Store

- Register rt is passed on busB into memory
- Memory address calculated just as in `lw` case

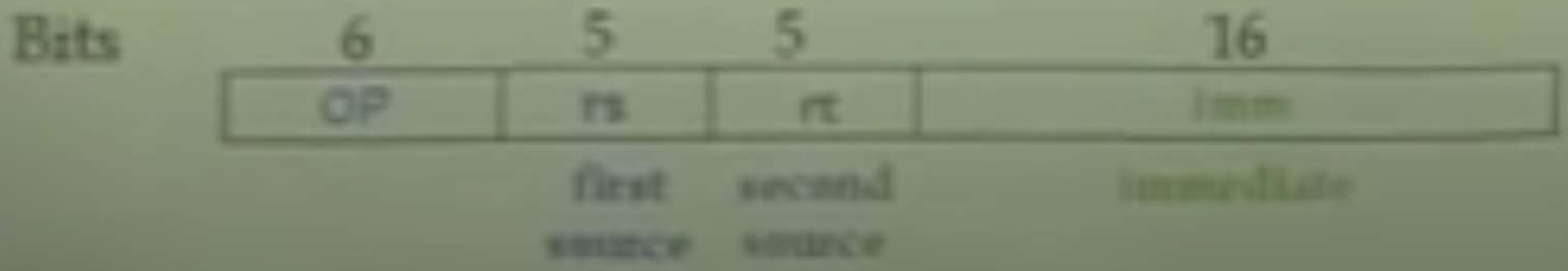


Branch RTL

- Branch instruction

```
beq rs, rt, imm

Inst[PC];                          Fetch instruction from memory
Cond <- R[rs] - R[rt];             Calculate branch condition
if (Cond eq 0)                     Test if equal
      PC <- PC + 4 +
                  SignExt(imm)*4;  Calculate PC Relative address
else
      PC <- PC + 4;                Calculate next address
```



Datapath: Branch