Computer Architecture
（计算机体系结构）

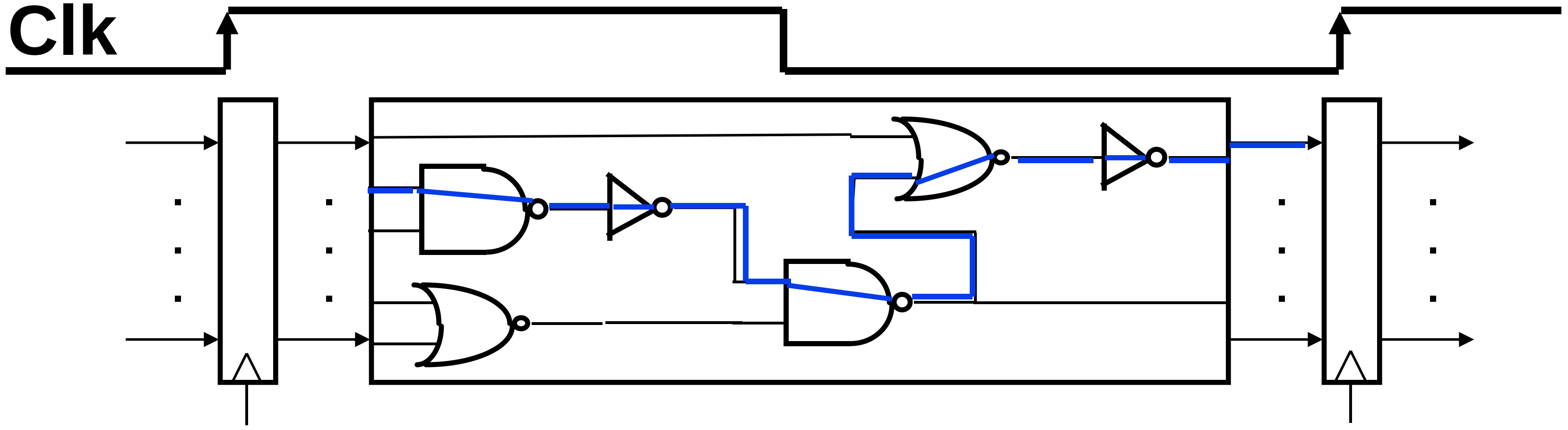
Lecture 21
CPU Design: Designing a Single-cycle CPU, pt 2

2020-10-19

How to Design a Processor: step-by-step

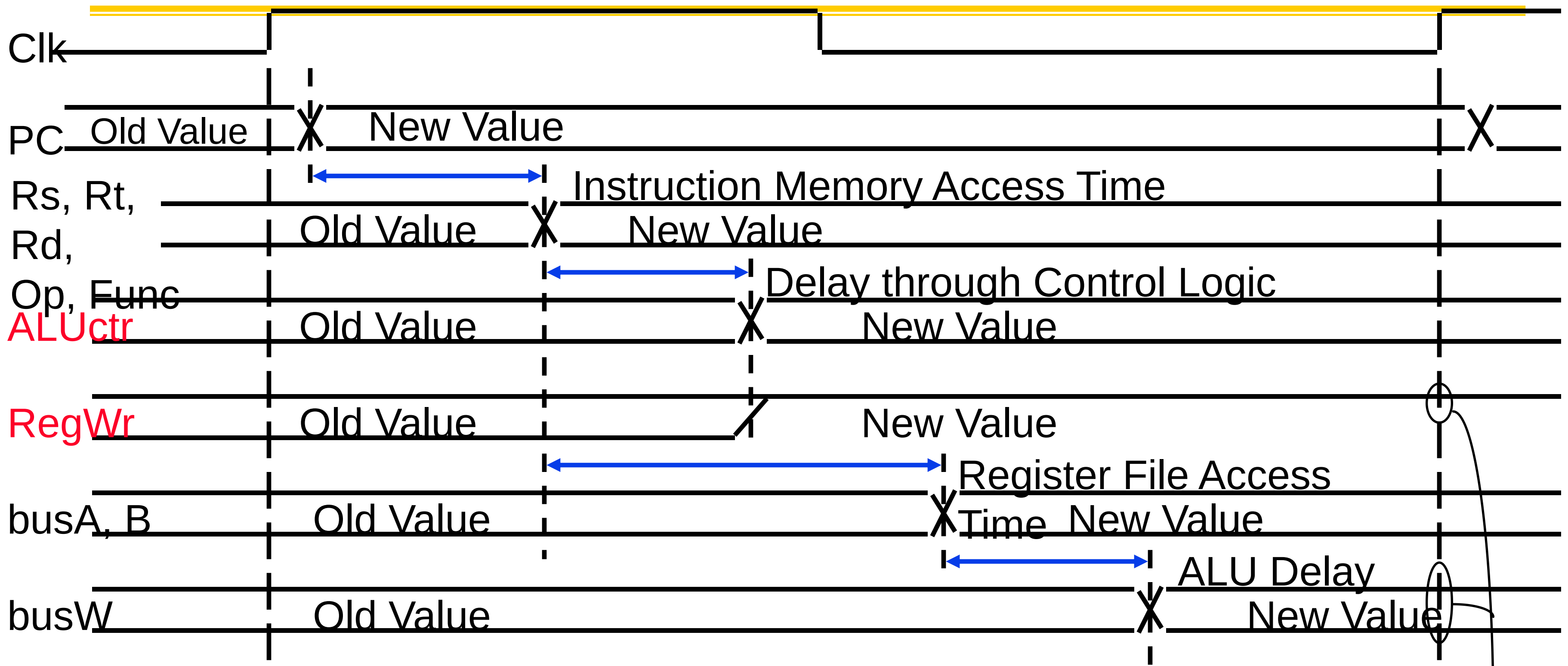
1. Analyze instruction set architecture (ISA) => datapath requirements
    - meaning of each instruction is given by the *register transfers*
    - datapath must include storage element for ISA registers
    - datapath must support each register transfer
2. Select set of datapath components and establish clocking methodology
3. Assemble datapath meeting requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic

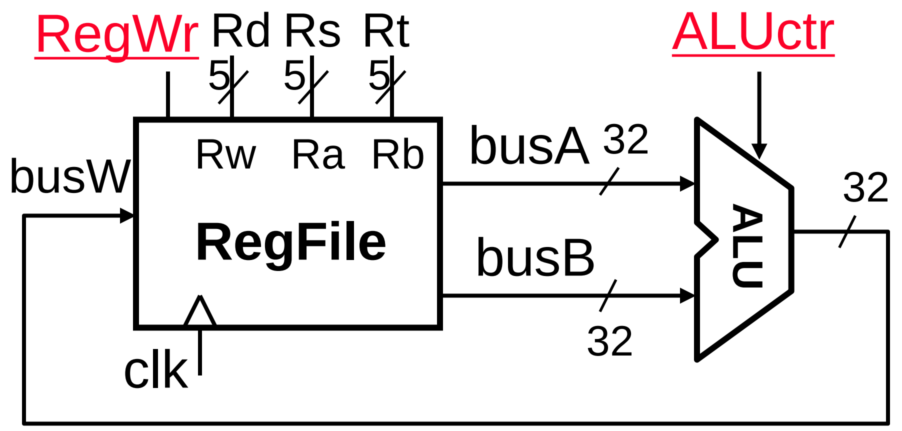
Clocking Methodology



- Storage elements clocked by same edge
- Being physical devices, flip-flops (FF) and combinational logic have some delays
  - Gates: delay from input change to output change
  - Signals at FF D input must be stable before active clock edge to allow signal to travel within the FF (set-up time), and we have the usual clock-to-Q delay
- "Critical path" (longest path through logic) determines length of clock period

Register-Register Timing: One complete cycle

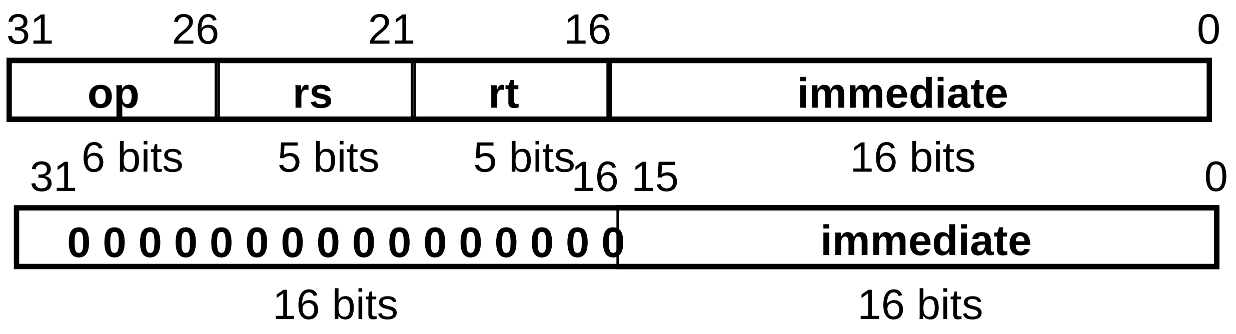




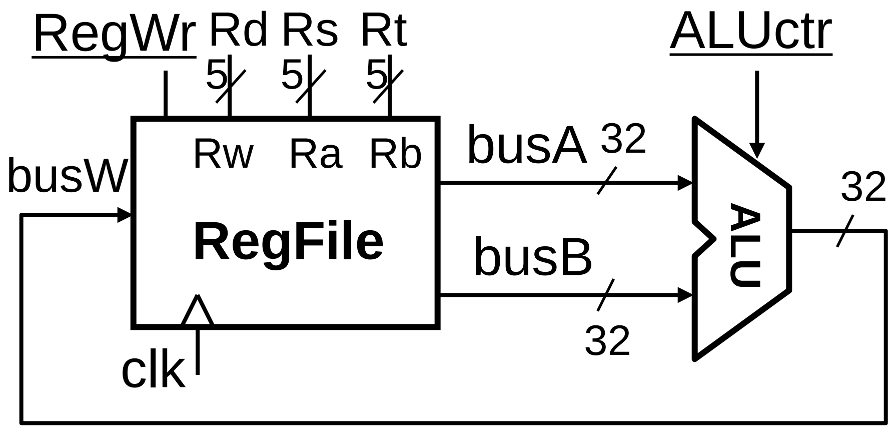
Register Write Occurs Here

3c: Logical Operations with Immediate

- R[rt] = R[rs] op ZeroExt[imm16]

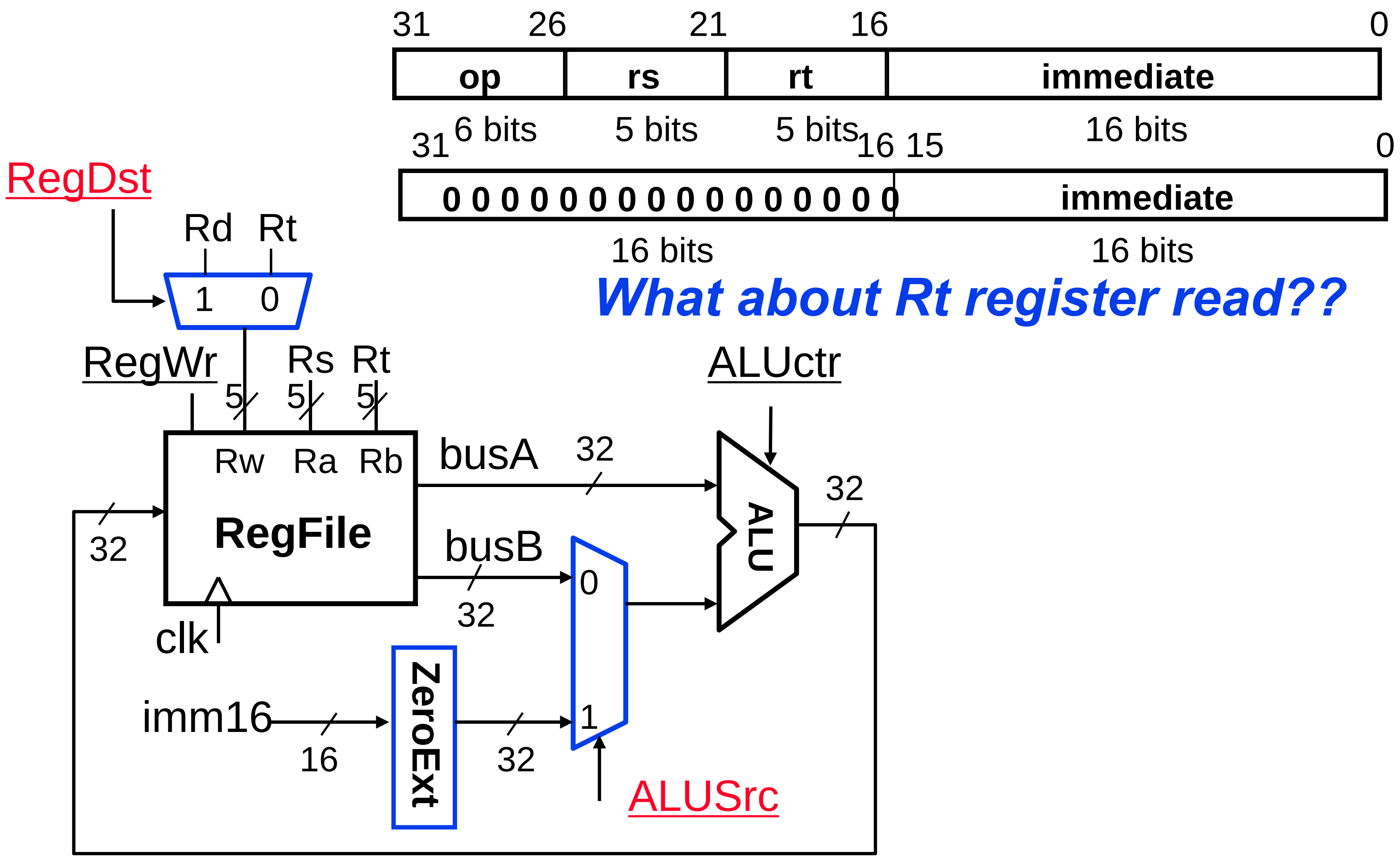


But we’re writing to Rt register??



3c: Logical Operations with Immediate

- R[rt] = R[rs] op ZeroExt[imm16]]

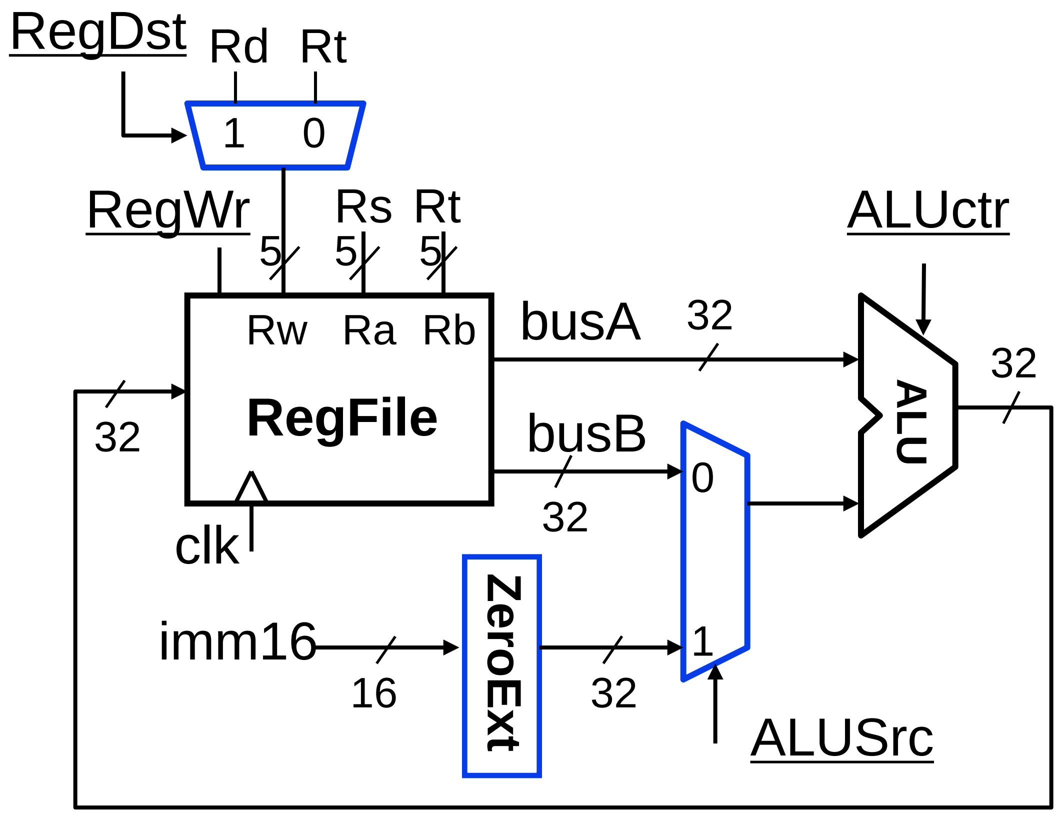


- Already defined 32-bit MUX; Zero Ext?

3d: Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`

31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	

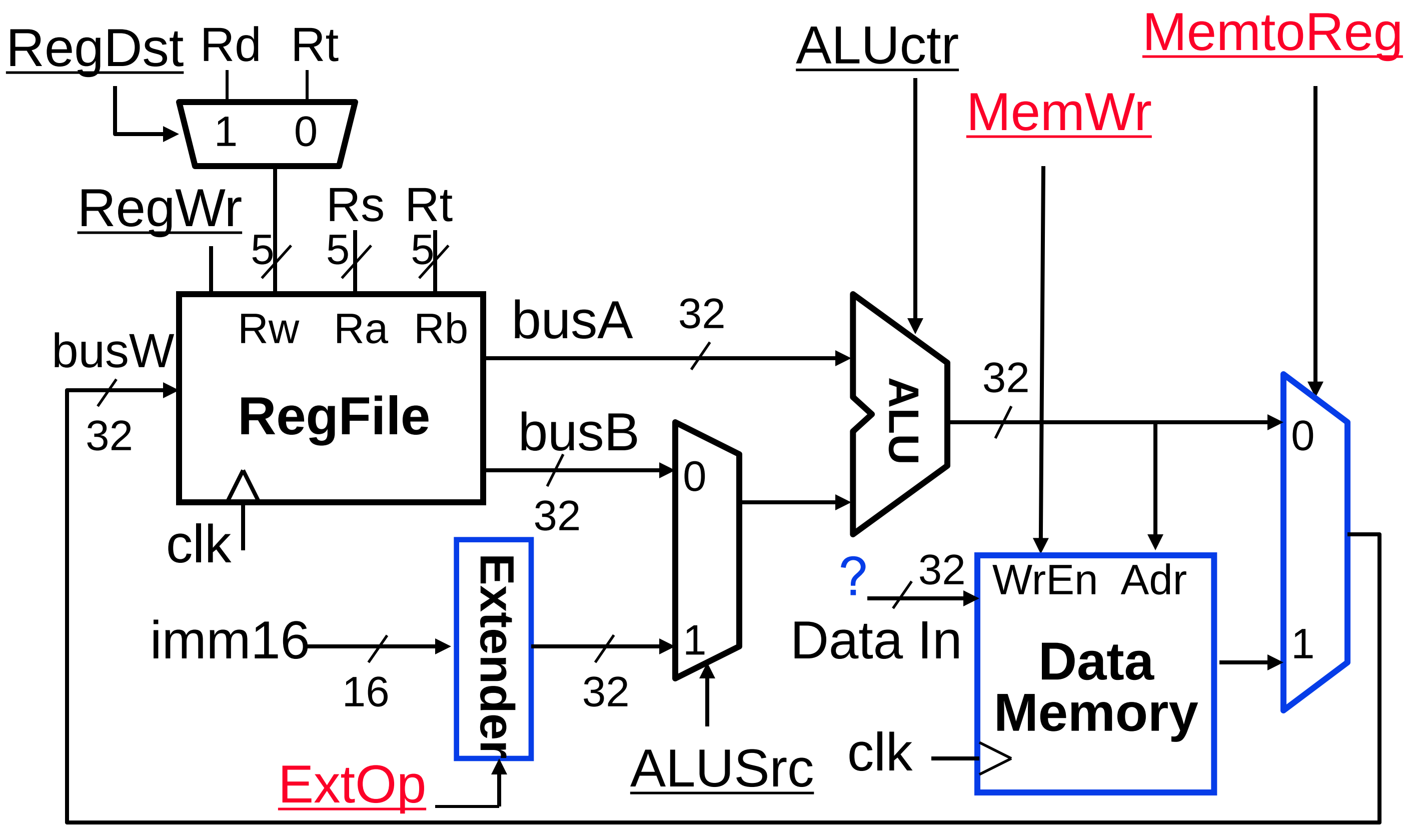


3d: Load Operations

- R[rt] = Mem[R[rs] + SignExt[imm16]]

  Example: `lw rt,rs,imm16`

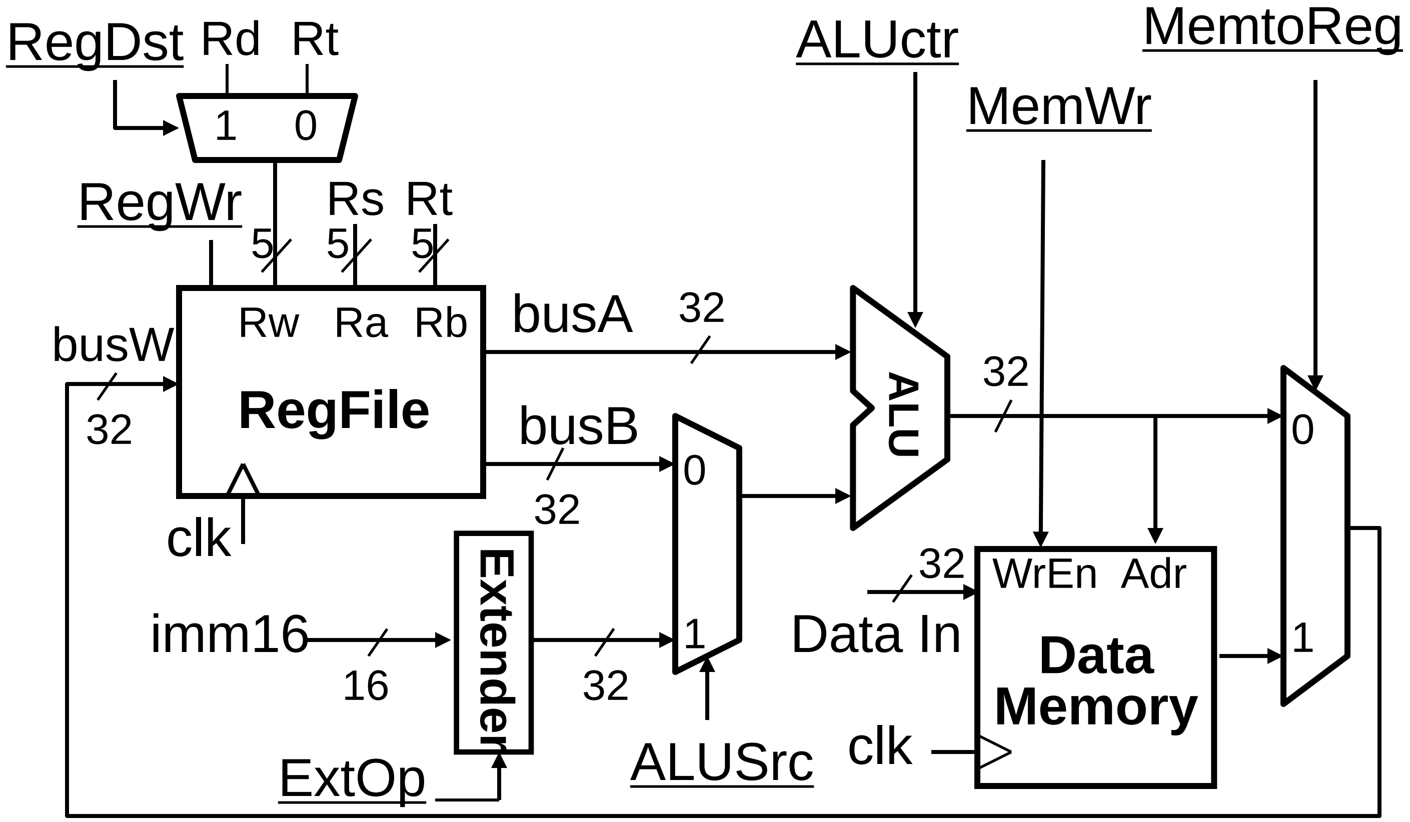
31	26	21	16 … 0
op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits



3e: Store Operations

- **Mem[R[rs] + SignExt[imm16]] = R[rt]**
  Ex.: `sw rt, rs, imm16`

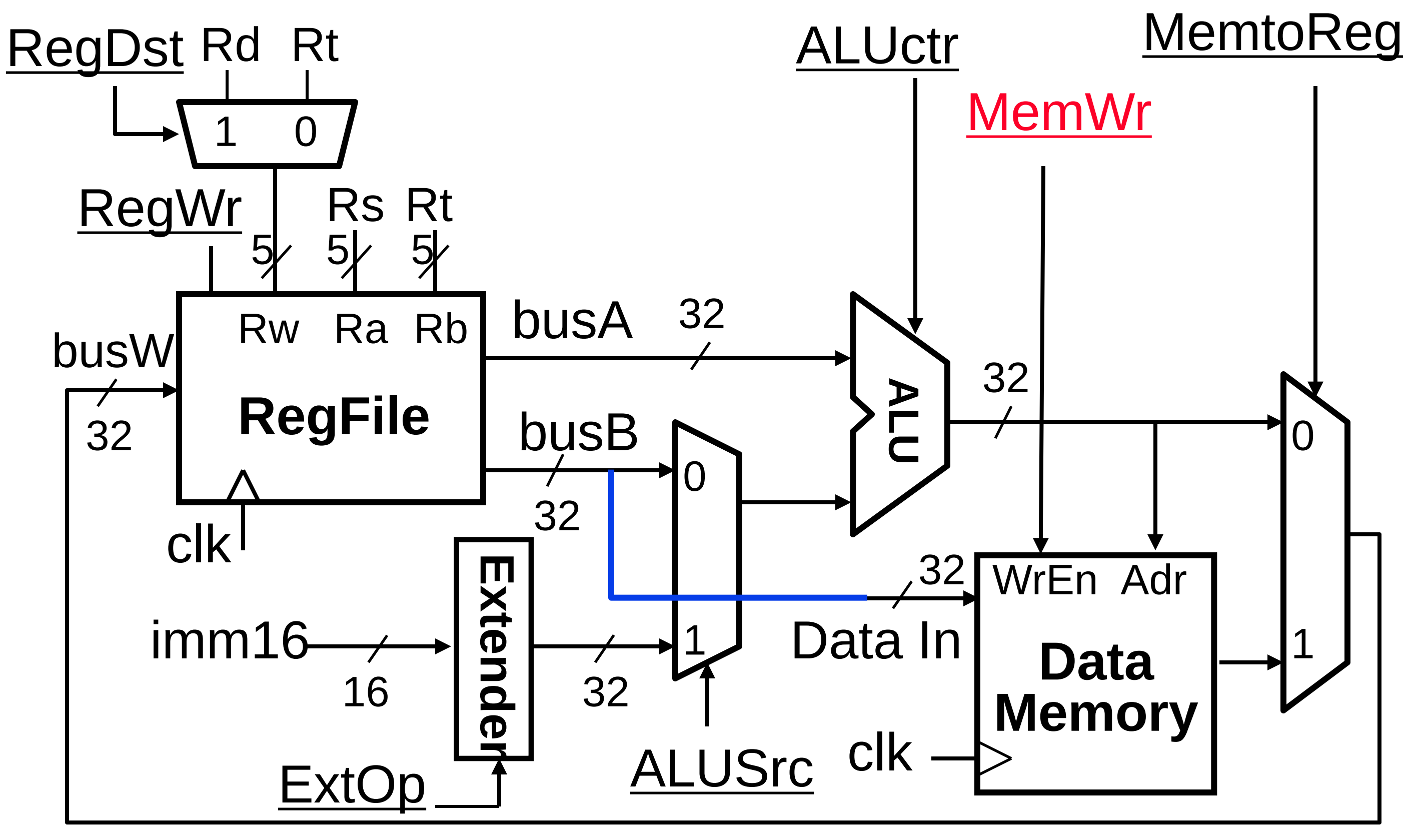
31–26	25–21	20–16	15–0
op	**rs**	**rt**	**immediate**
6 bits	5 bits	5 bits	16 bits



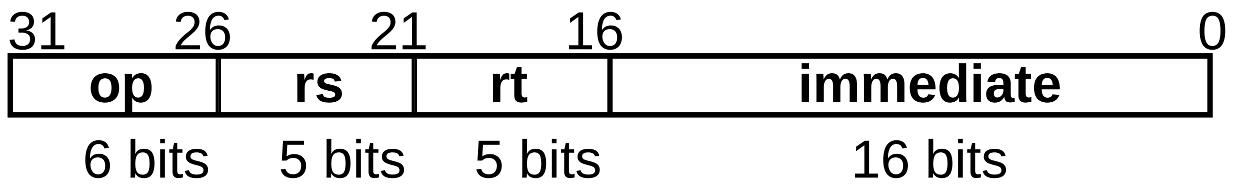
3e: Store Operations

- **Mem[R[rs] + SignExt[imm16]] = R[rt]**
  Ex.: `sw rt, rs, imm16`

31	26	21	16	0
op	**rs**	**rt**	**immediate**	
6 bits	5 bits	5 bits	16 bits	



3f: The Branch Instruction



```
beq rs, rt, imm16
```

- mem[PC] Fetch the instruction from memory
- Equal = R[rs] == R[rt] Calculate branch condition
- if (Equal) Calculate the next instruction's address
  - PC = PC + 4 + (SignExt(imm16) x 4)

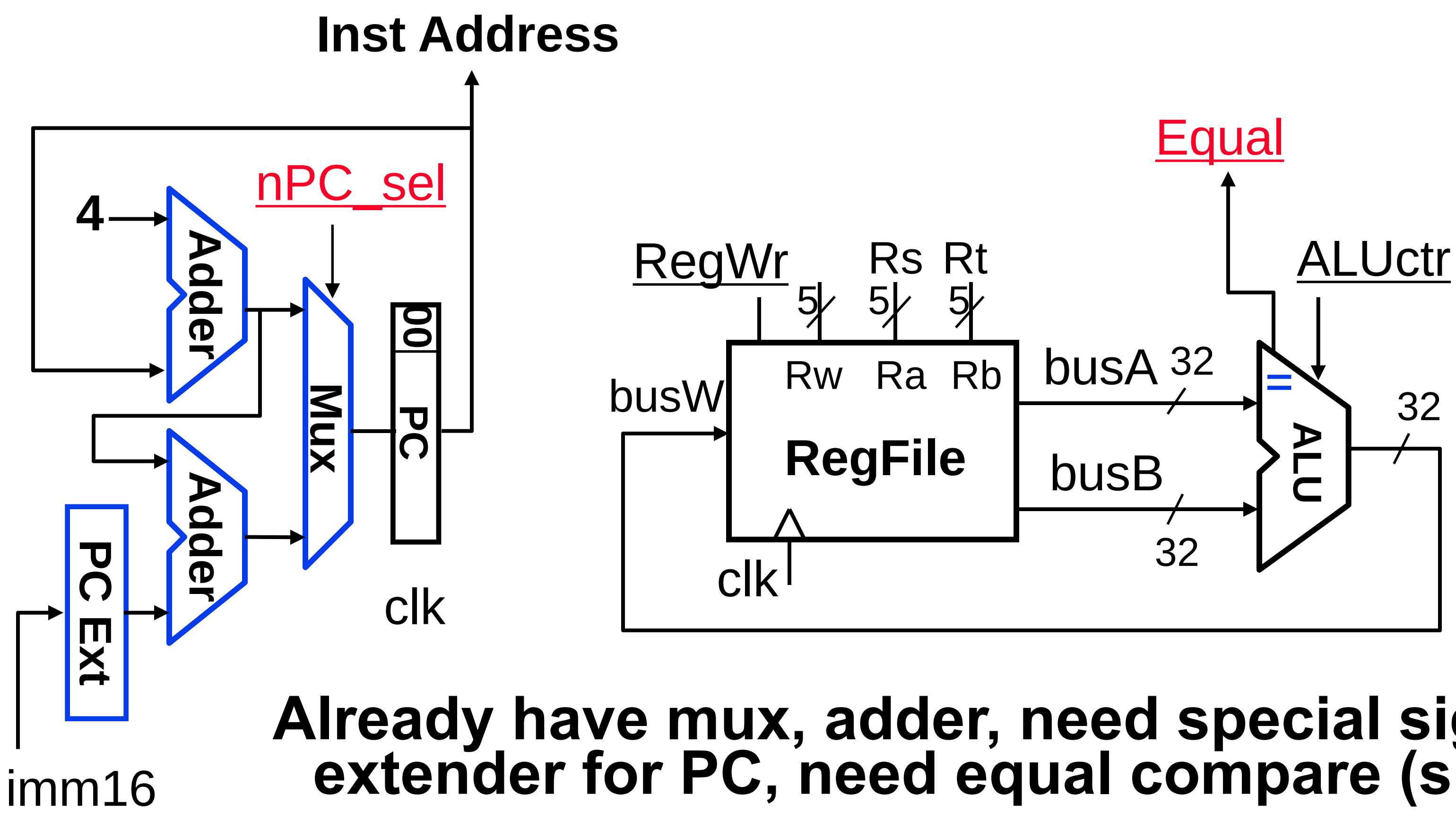
  else
  - PC = PC + 4

Datapath for Branch Operations

- **beq rs, rt, imm16**
  Datapath generates condition (equal)

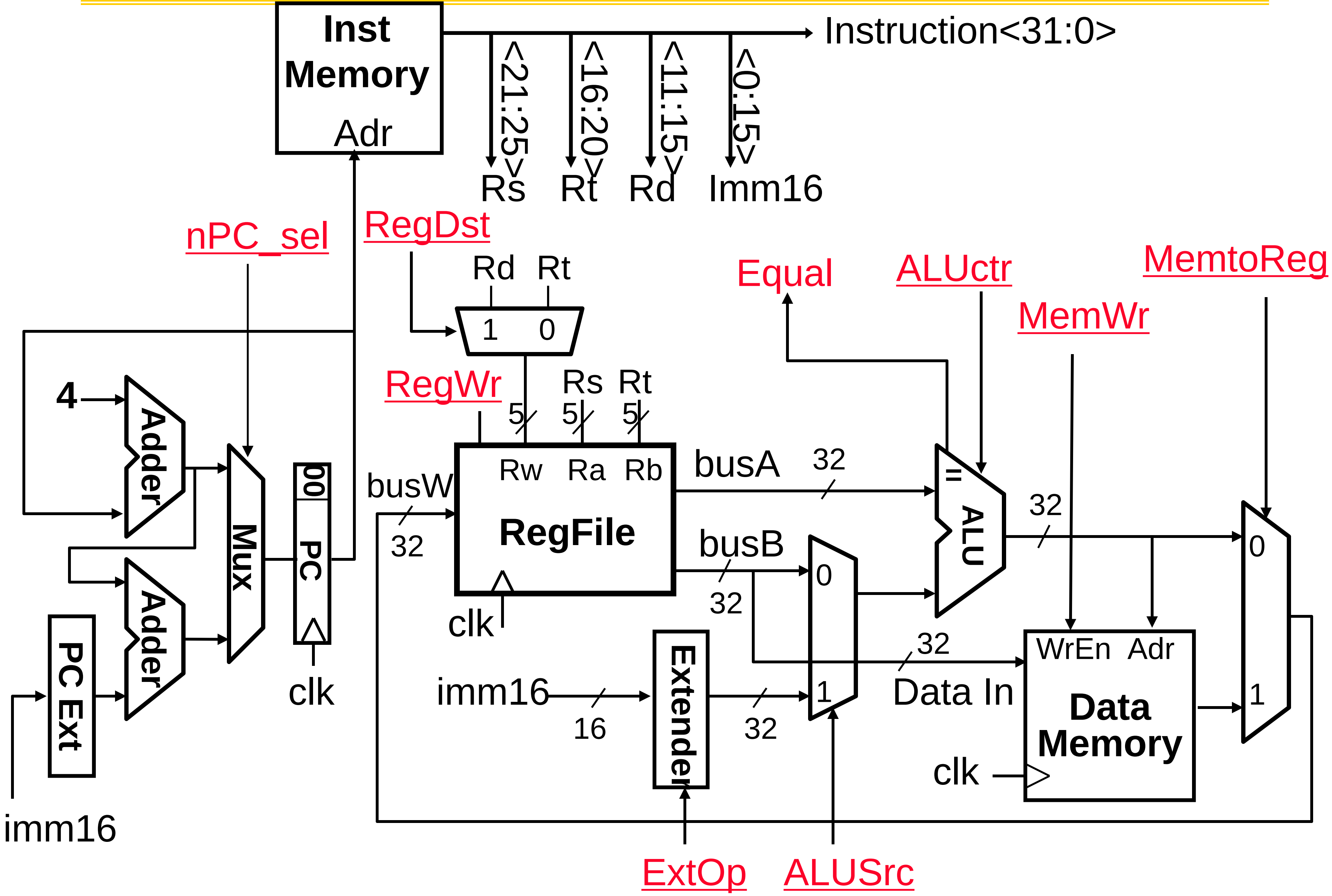
31 26 21 16 0

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits

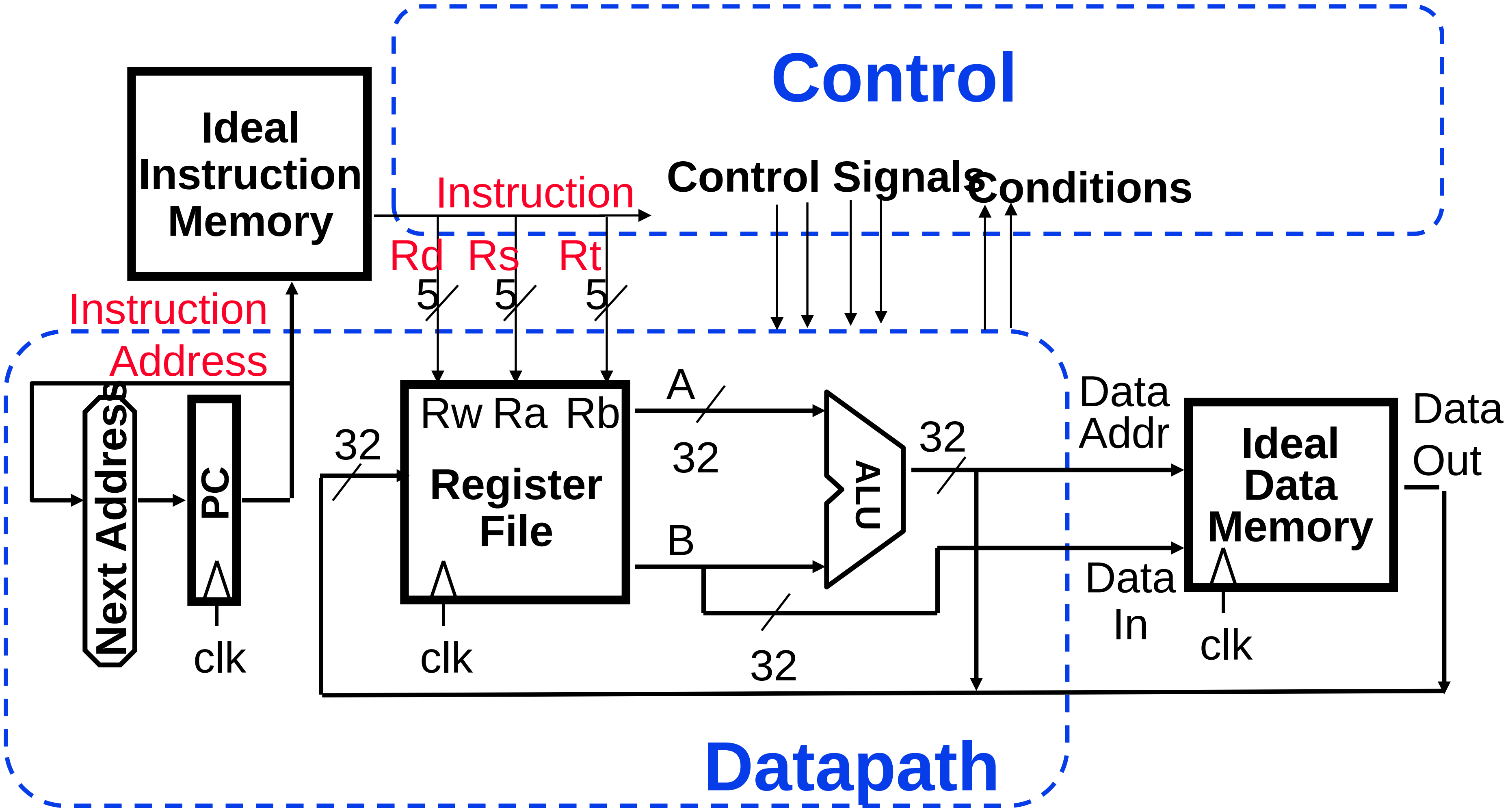


Already have mux, adder, need special sign extender for PC, need equal compare (sub?)

Putting it All Together:A Single Cycle Datapath



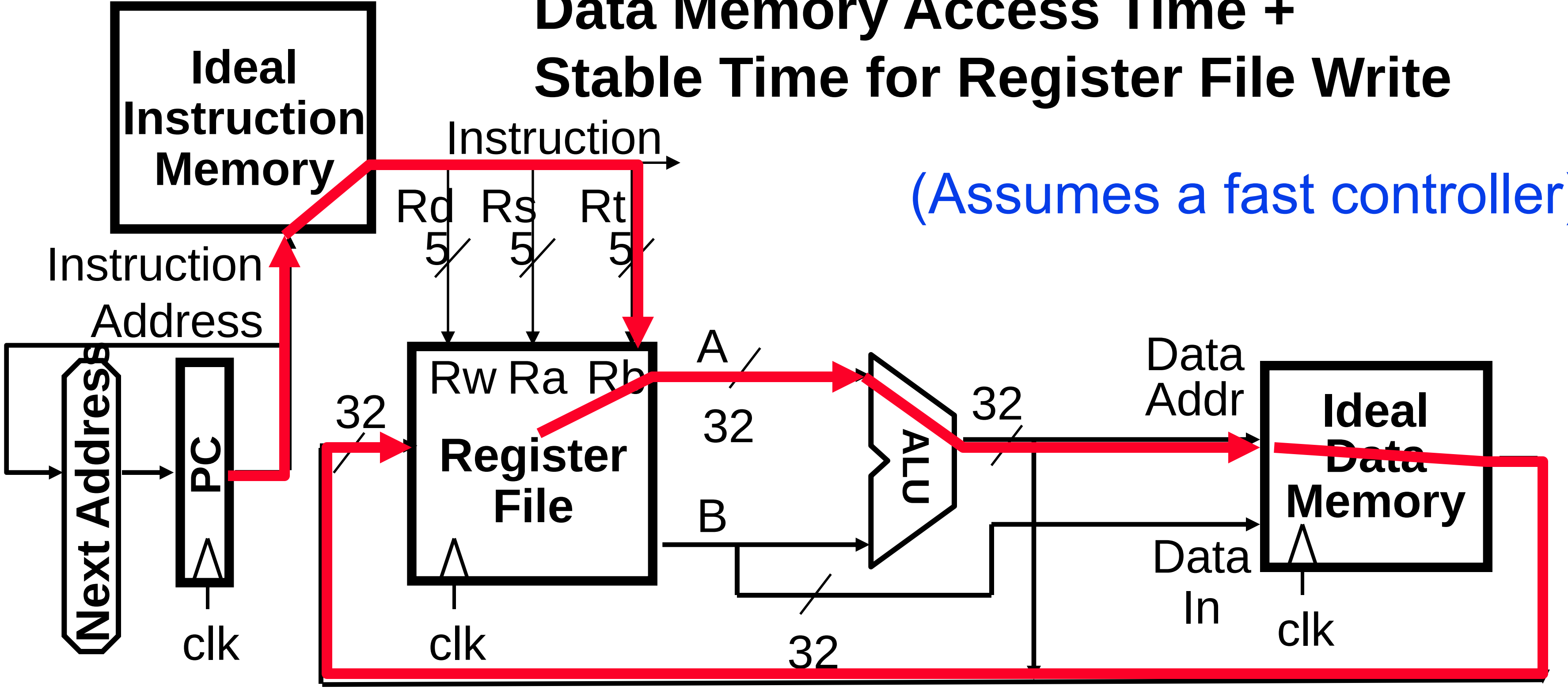
An Abstract View of the Implementation



An Abstract View of the Critical Path

Critical Path (Load Instruction) =
- **Delay clock through PC (FFs) +**
- **Instruction Memory's Access Time +**
- **Register File's Access Time, +**
- **ALU to Perform a 32-bit Add +**
- **Data Memory Access Time +**
- **Stable Time for Register File Write**

(Assumes a fast controller)



Peer Instruction

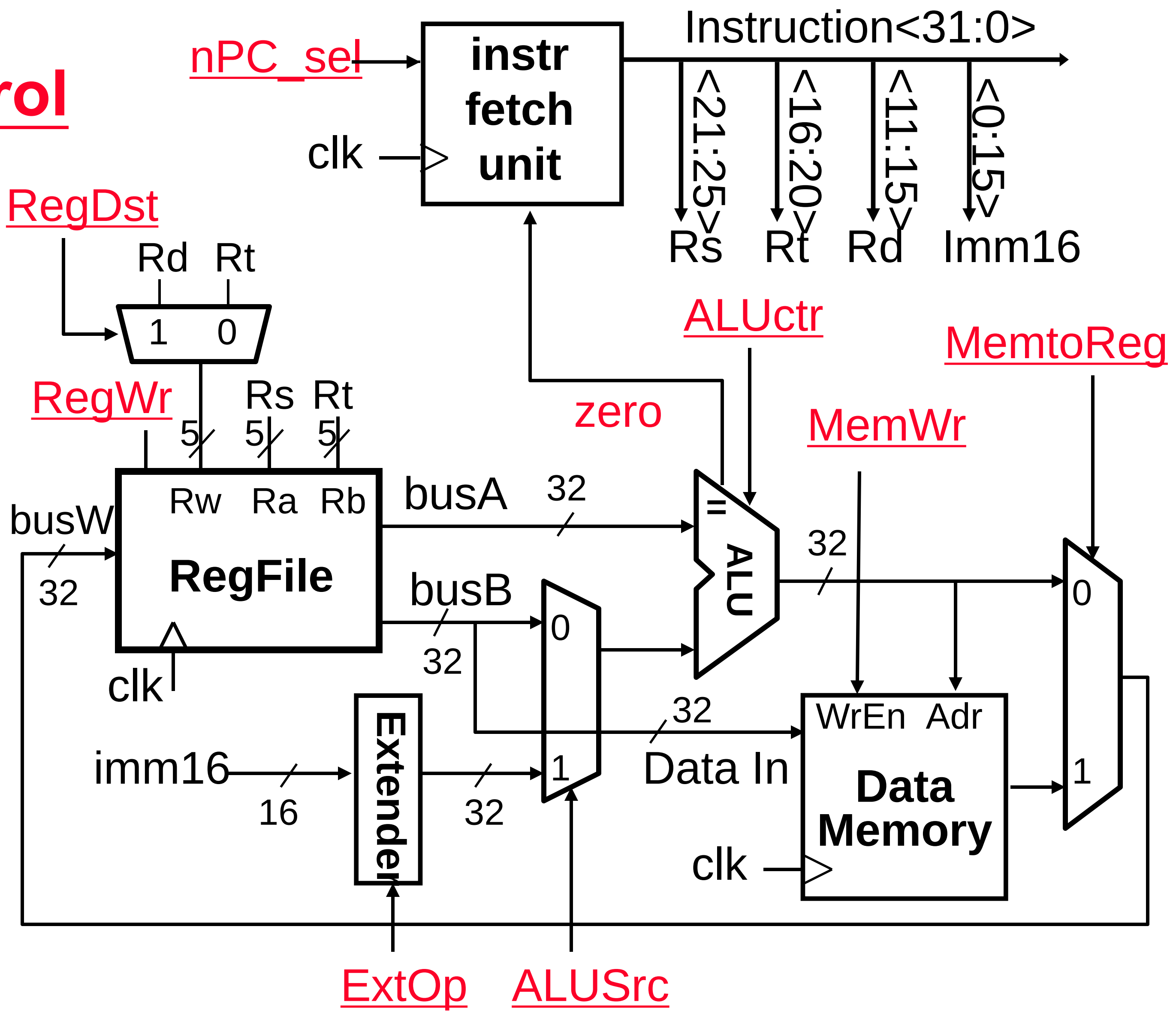
1) In the worst case, the delay is the memory access time

2) With <u>only</u> changes to control, our datapath could write to memory <u>and</u> registers in one cycle.

	12
a)	FF
b)	FT
c)	TF
d)	TT

Summary: A Single Cycle Datapath

- **We have everything except control signals**



nPC_sel
clk
instr fetch unit
Instruction<31:0>
<21:25>
<16:20>
<11:15>
<0:15>
Rs
Rt
Rd
Imm16
RegDst
Rd
Rt
1
0
ALUctr
MemtoReg
RegWr
Rs
Rt
zero
MemWr
5
5
5
Rw
Ra
Rb
busA
32
busW
RegFile
=
ALU
32
32
busB
0
0
clk
32
32
WrEn
Adr
imm16
Extender
1
Data In
Data Memory
1
16
32
clk
ExtOp
ALUSrc