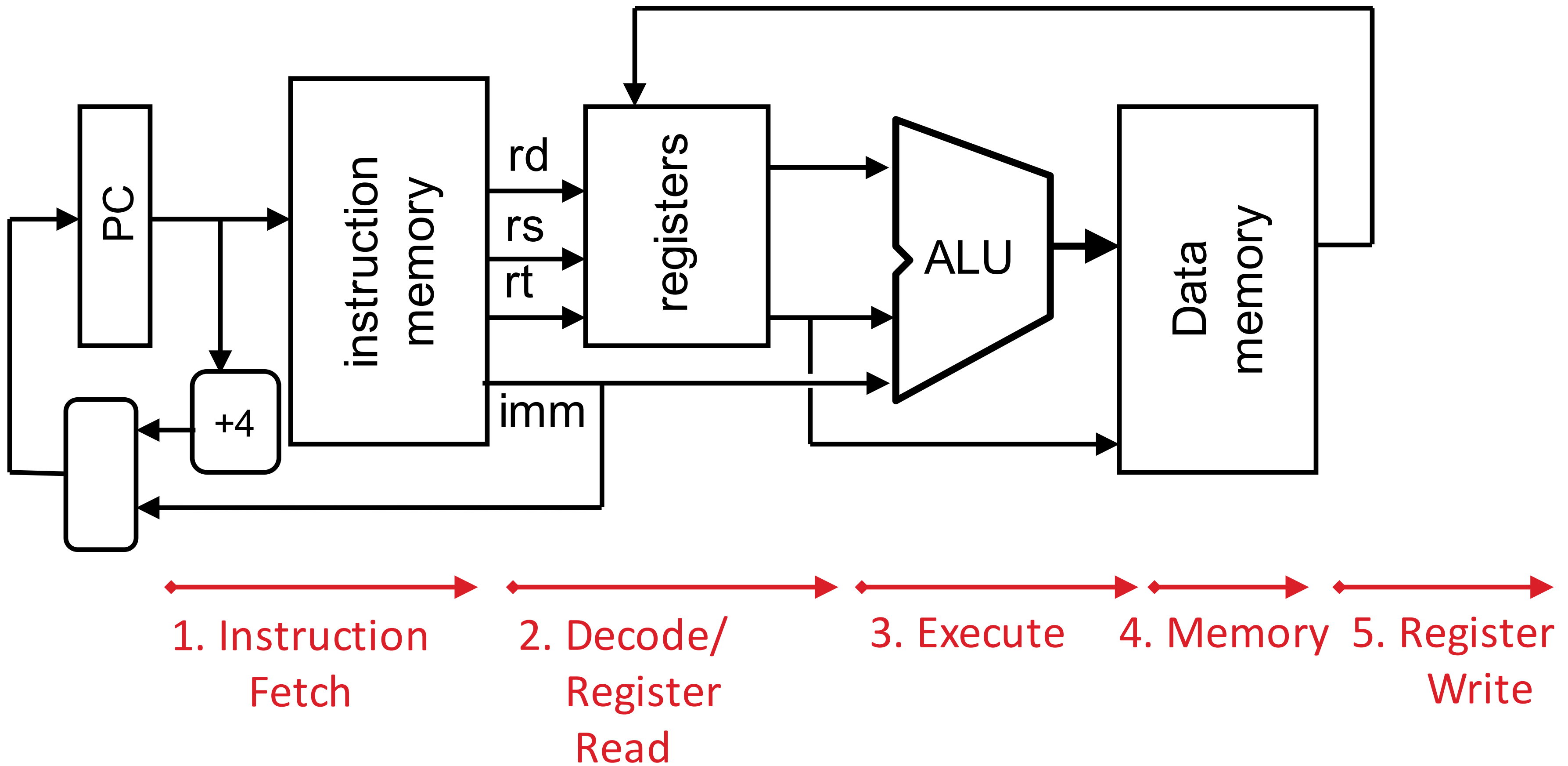
CS 61C:
Great Ideas in Computer Architecture
Single-Cycle CPU
Datapath & Control

Instructors:

Vladimir Stojanovic and Nicholas Weaver

http://inst.eecs.Berkeley.edu/~cs61c/sp16

Stages of Execution on Datapath



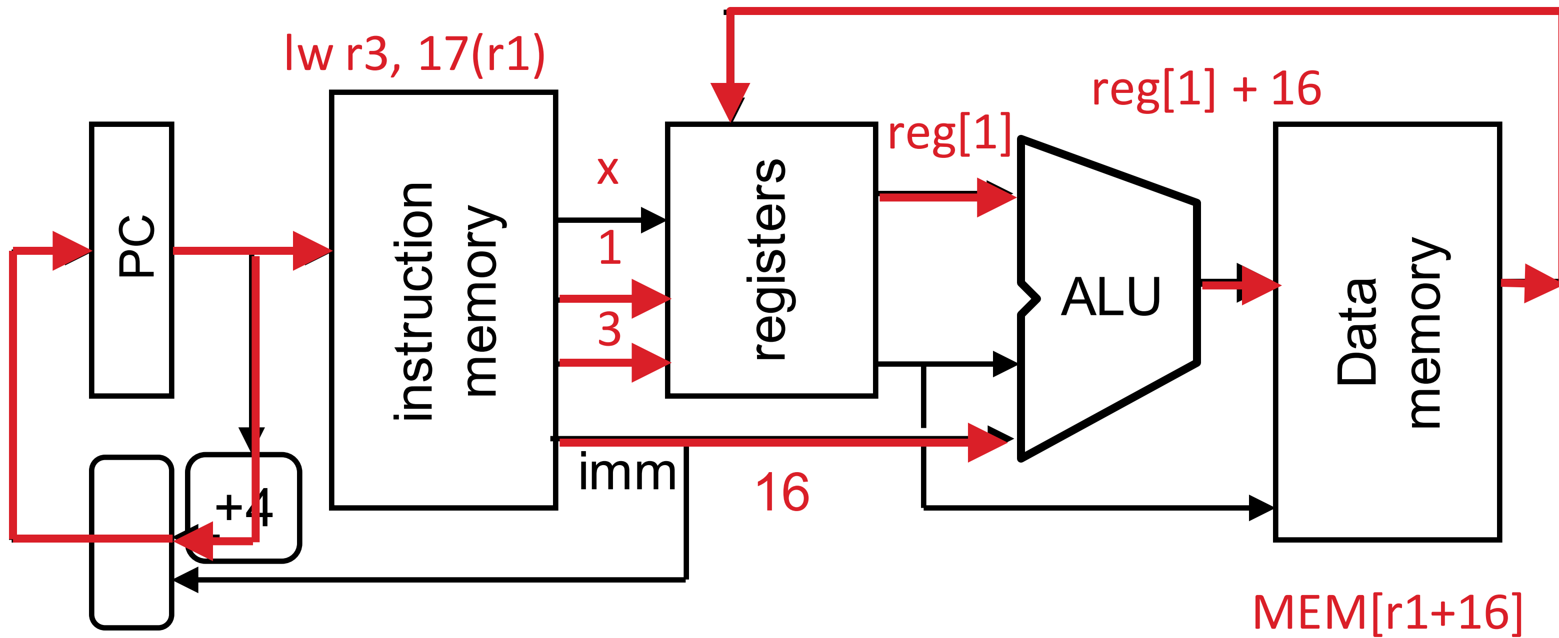
Why Five Stages? (1/2)

- Could we have a different number of stages?
  - Yes, other ISAs have different natural number of stages
    - And these days, pipelining can be much more aggressive than the "natural" 5 stages MIPS uses
- Why does MIPS have five if instructions tend to idle for at least one stage?
  - Five stages are the union of all the operations needed by all the instructions.
  - One instruction uses all five stages: the load

Why Five Stages? (2/2)

- lw $r3,16($r1) # r3=Mem[r1+16]
  - Stage 1: fetch this instruction, increment PC
  - Stage 2: decode to determine it is a lw, then read register $r1
  - Stage 3: add 16 to value in register $r1 (retrieved in Stage 2)
  - Stage 4: read value from memory address computed in Stage 3
  - Stage 5: write value read in Stage 4 into register $r3

Example: lw Instruction



Clickers/Peer Instruction

- Which type of MIPS instruction is active in the fewest stages?

A: LW

B: BEQ

C: J

D: JAL

E: ADDU

Processor Design: 5 steps

Step 1: Analyze instruction set to determine datapath requirements

- Meaning of each instruction is given by register transfers
- Datapath must include storage element for ISA registers
- Datapath must support each register transfer

Step 2: Select set of datapath components & establish clock methodology

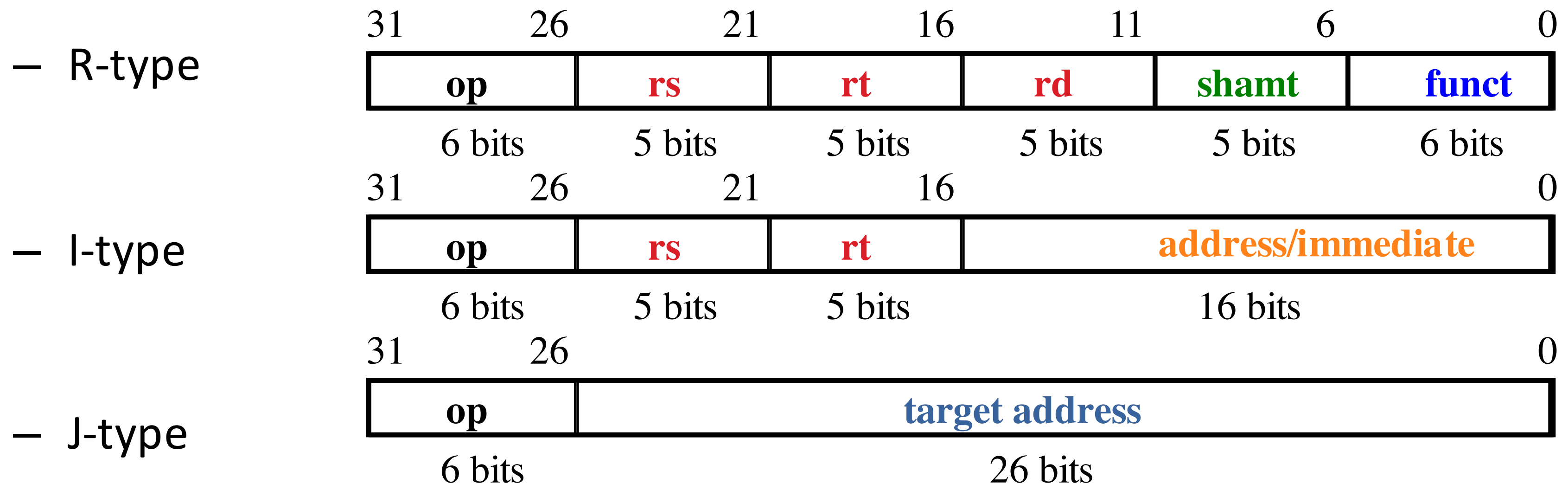
Step 3: Assemble datapath components that meet the requirements

Step 4: Analyze implementation of each instruction to determine setting of control points that realizes the register transfer

Step 5: Assemble the control logic

The MIPS Instruction Formats

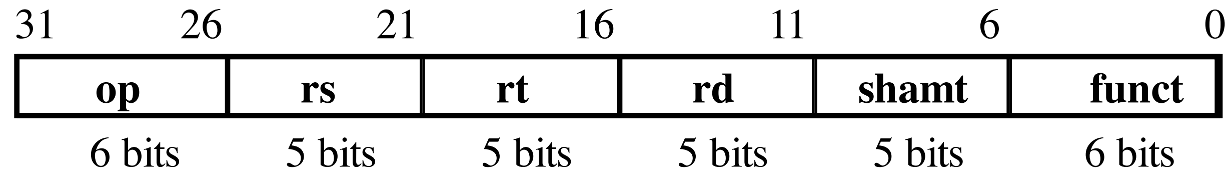
- All MIPS instructions are 32 bits long. 3 formats:
  - R-type
  - I-type
  - J-type



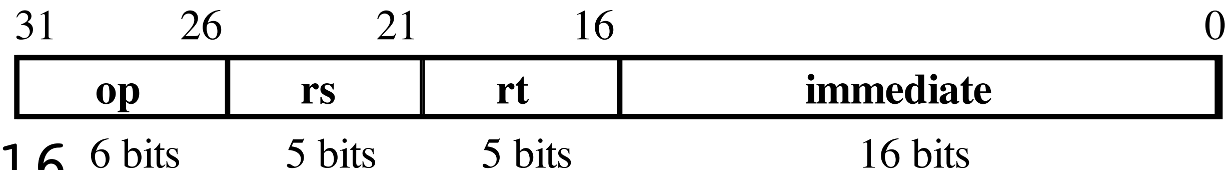
- The different fields are:
  - op: operation ("opcode") of the instruction
  - rs, rt, rd: the source and destination register specifiers
  - shamt: shift amount
  - funct: selects the variant of the operation in the "op" field
  - address / immediate: address offset or immediate value
  - target address: target address of jump instruction

The MIPS-lite Subset

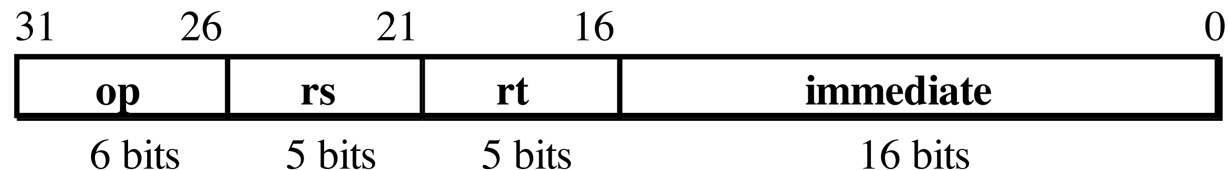
- ADDU and SUBU
  - `addu rd,rs,rt`
  - `subu rd,rs,rt`



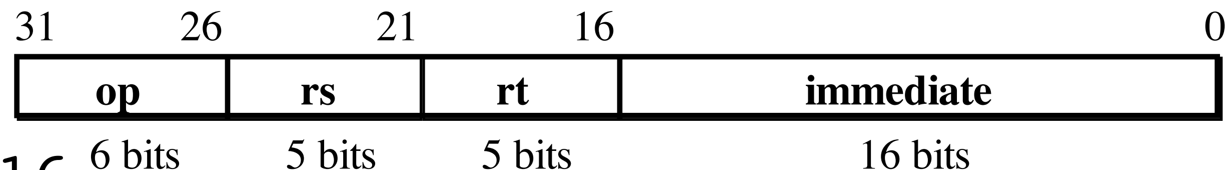
- OR Immediate:
  - `ori rt,rs,imm16`



- LOAD and STORE Word
  - `lw rt,rs,imm16`
  - `sw rt,rs,imm16`



- BRANCH:
  - `beq rs,rt,imm16`



Register Transfer Level (RTL)

- Colloquially called "Register Transfer Language"
- RTL gives the meaning of the instructions
- All start by fetching the instruction itself

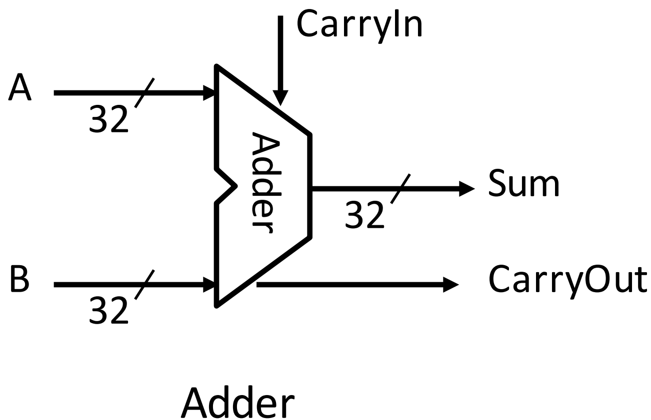
```
{op , rs , rt , rd , shamt , funct} ← MEM[ PC ]

{op , rs , rt ,   Imm16} ← MEM[ PC ]

Inst   Register Transfers

ADDU   R[rd] ← R[rs] + R[rt]; PC ← PC + 4

SUBU   R[rd] ← R[rs] – R[rt]; PC ← PC + 4

ORI    R[rt] ← R[rs] | zero_ext(Imm16); PC ← PC + 4

LOAD   R[rt] ← MEM[ R[rs] + sign_ext(Imm16)]; PC ← PC + 4

STORE  MEM[ R[rs] + sign_ext(Imm16) ] ← R[rt]; PC ← PC + 4

BEQ    if ( R[rs] == R[rt] )
           PC ← PC + 4 + {sign_ext(Imm16), 2'b00}
       else PC ← PC + 4
```

Step 1: Requirements of the Instruction Set

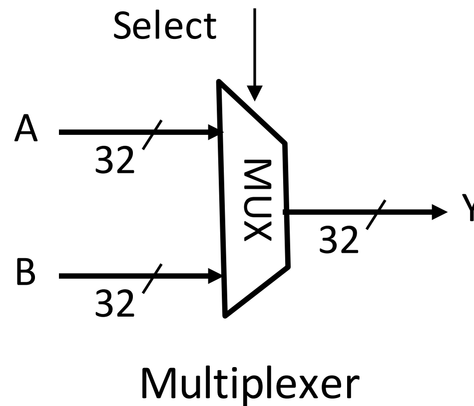
- Memory (MEM)
  - Instructions & data (will use one for each)
- Registers (R: 32, 32-bit wide registers)
  - Read RS
  - Read RT
  - Write RT or RD
- Program Counter (PC)
- Extender (sign/zero extend)
- Add/Sub/OR/etc unit for operation on register(s) or extended immediate (ALU)
- Add 4 (+ maybe extended immediate) to PC
- Compare registers?

Step 2: Components of the Datapath

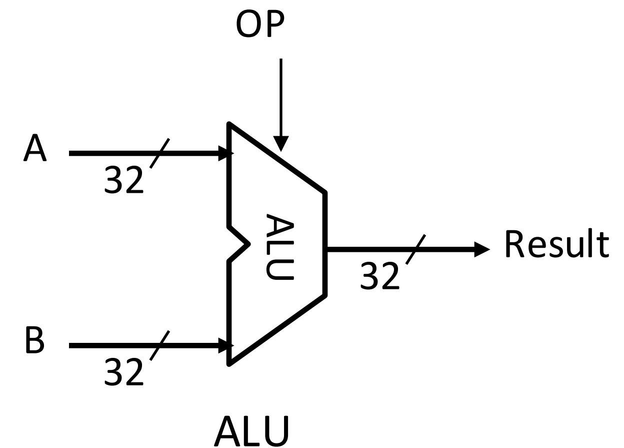
- Combinational Elements
- Storage Elements + Clocking Methodology
- Building Blocks



Adder



Multiplexer



ALU

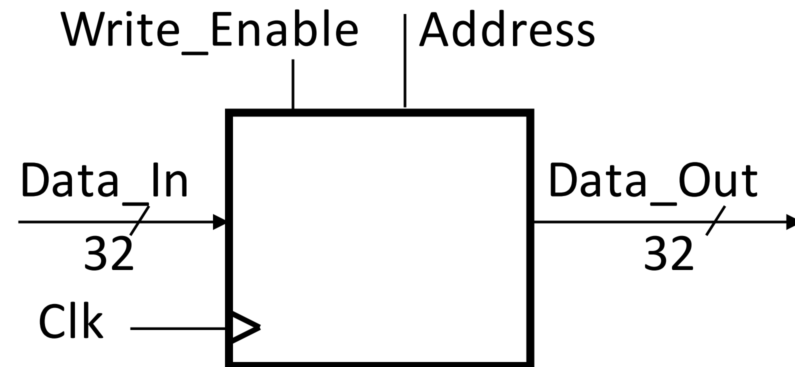
ALU Needs for MIPS-lite + Rest of MIPS

- Addition, subtraction, logical OR, ==:

```
ADDU   R[rd] = R[rs] + R[rt]; ...
SUBU   R[rd] = R[rs] – R[rt]; ...
ORI    R[rt] = R[rs] | zero_ext(Imm16)...
BEQ    if ( R[rs] == R[rt] )...
```

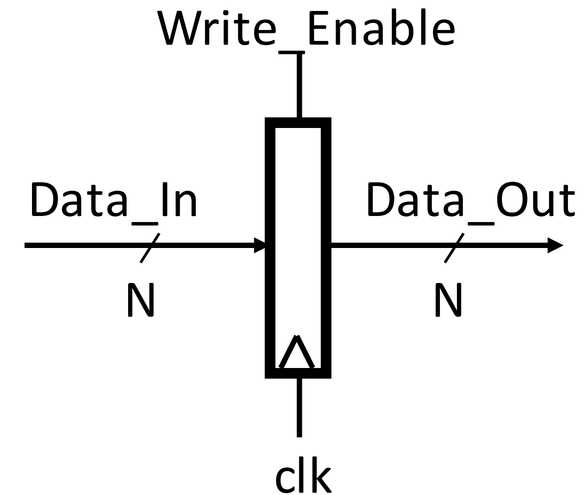
- Test to see if output == 0 for any ALU operation gives == test. How?
- P&H also adds AND, Set Less Than (1 if A < B, 0 otherwise)
- ALU follows Chapter 5

Storage Element: Idealized Memory



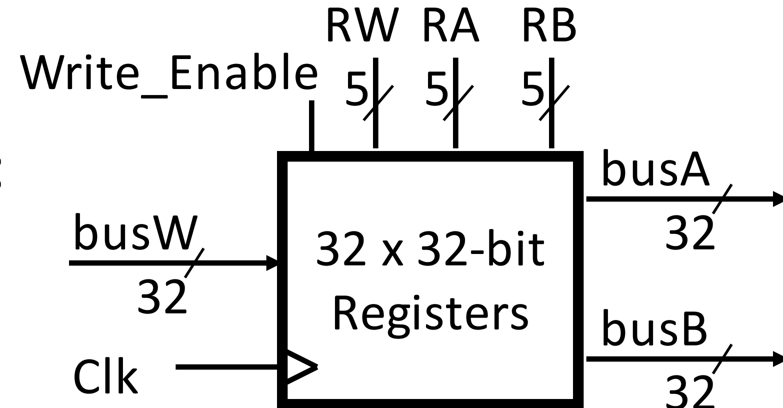
- "Magic" Memory
  - One input bus: Data In
  - One output bus: Data Out
- Memory word is found by:
  - For Read: Address selects the word to put on Data_Out
  - For Write: Set Write_Enable = 1: address selects the memory word to be written via the Data_In bus
- Clock input (CLK)
  - CLK input is a factor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

Storage Element: Register (Building Block)



- Similar to D Flip-Flop except
  - N-bit input and output
  - Write_Enable input
- Write_Enable:
  - Negated (or deasserted) (0): Data_Out will not change
  - Asserted (1): Data_Out will become Data_In on positive edge of clock

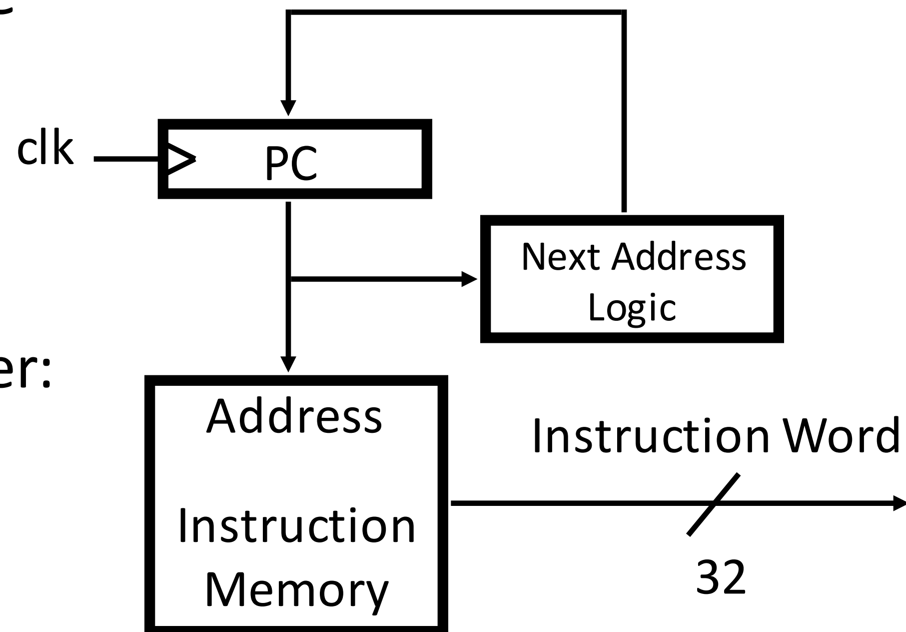
Storage Element: Register File



- Register File 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."

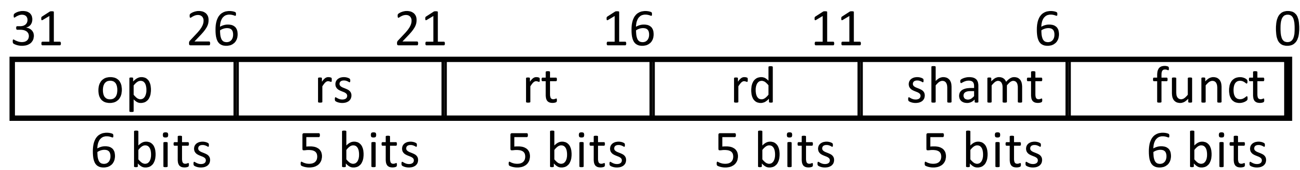
Step 3a: Instruction Fetch Unit

- Register Transfer Requirements ⇒ Datapath Assembly
- Instruction Fetch
- Read Operands and Execute Operation
- Common RTL operations
  - Fetch the Instruction: mem[PC]
  - Update the program counter:
    - Sequential Code: PC ← PC + 4
    - Branch and Jump: PC ← "something else"

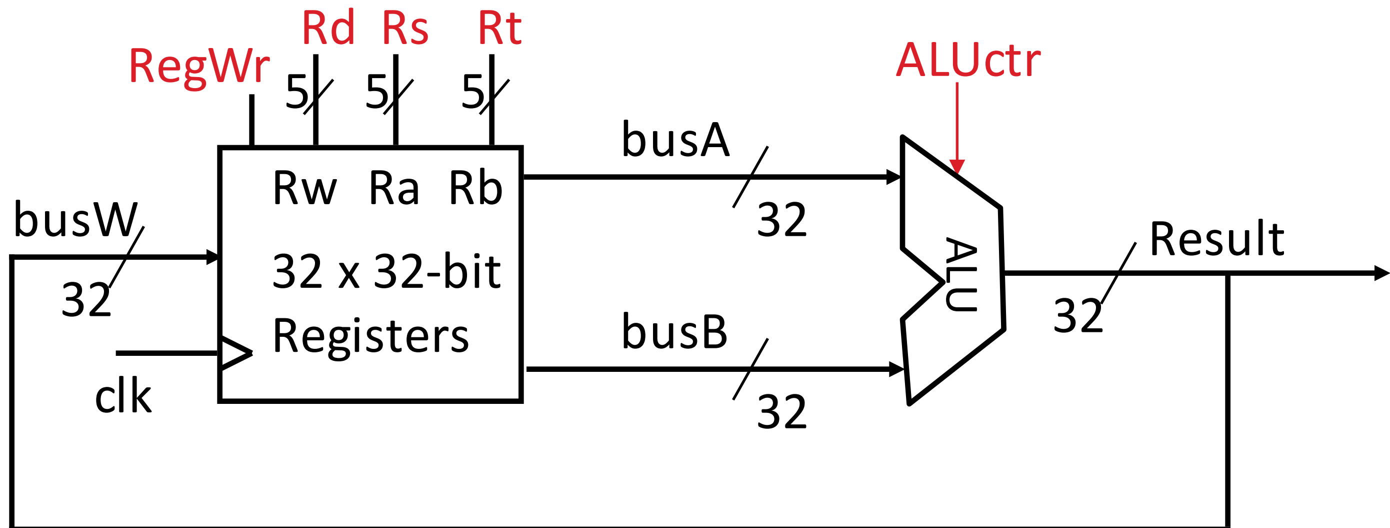


Step 3b: Add & Subtract

- R[rd] = R[rs] op R[rt] (addu rd,rs,rt)
  - Ra, Rb, and Rw come from instruction's Rs, Rt, and Rd fields

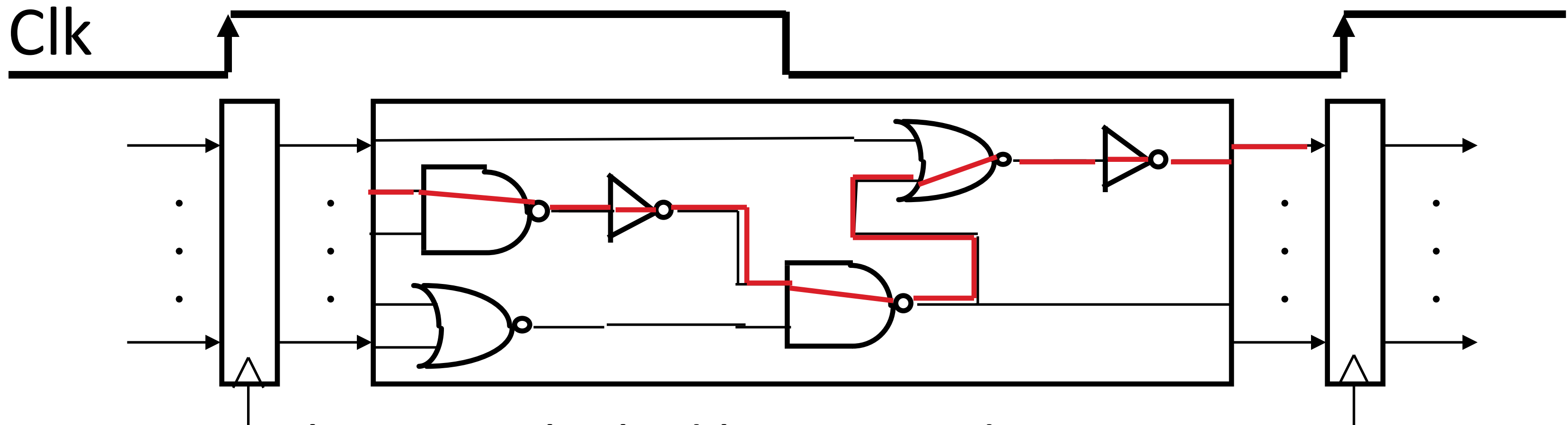


  - ALUctr and RegWr: control logic after decoding the instruction



- … Already defined the register file & ALU

Clocking Methodology



- Storage elements clocked by same edge
- Flip-flops (FFs) 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 (Add/Sub)

Clk

PC Old Value

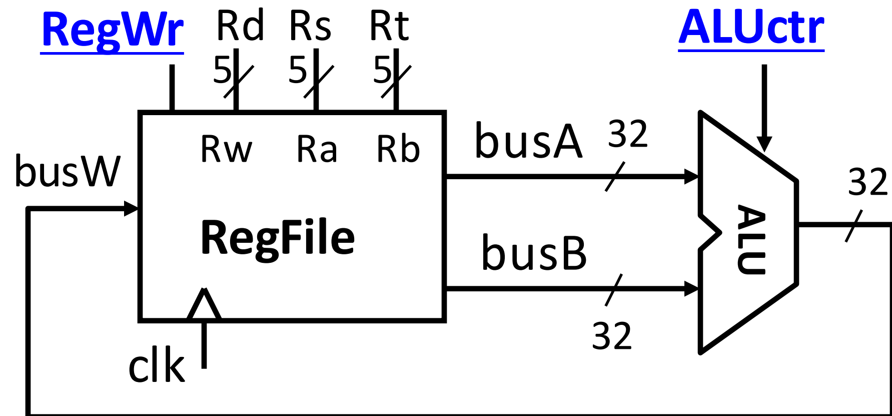
Rs, Rt, Rd, Op, Func

ALUctr

RegWr

busA, B

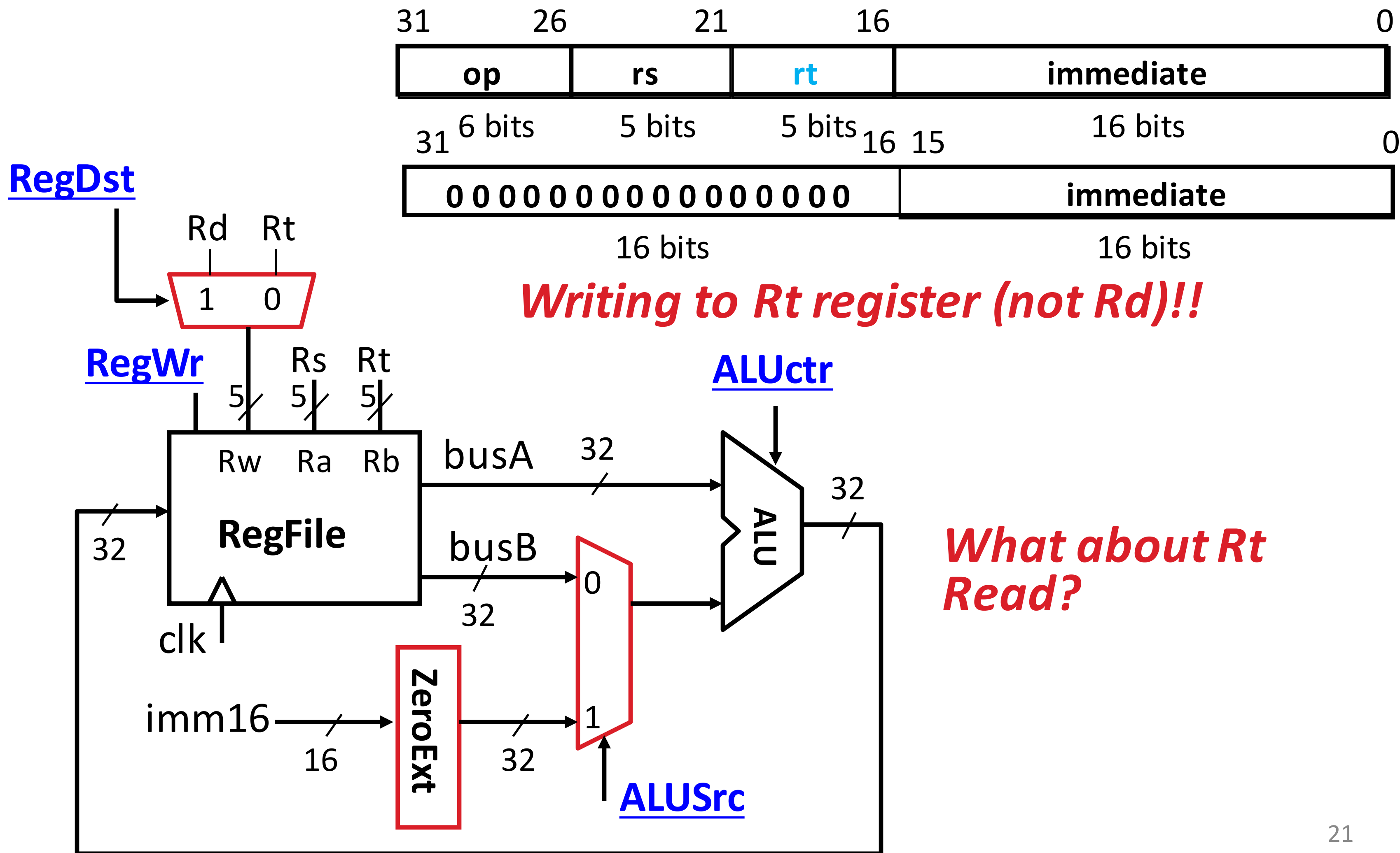
busW



Register Write Occurs Here

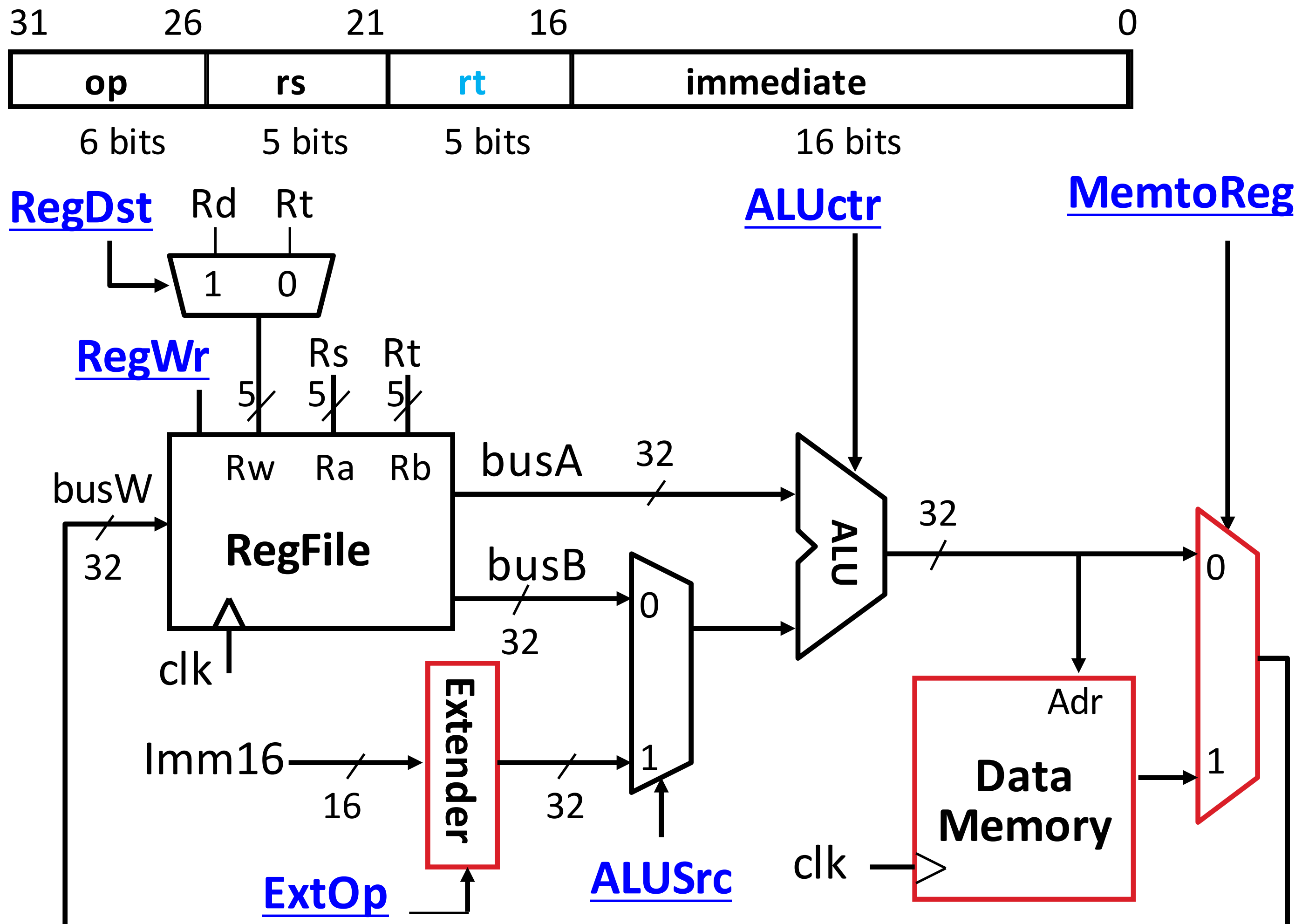
3c: Logical Op (or) with Immediate

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



3d: Load Operations

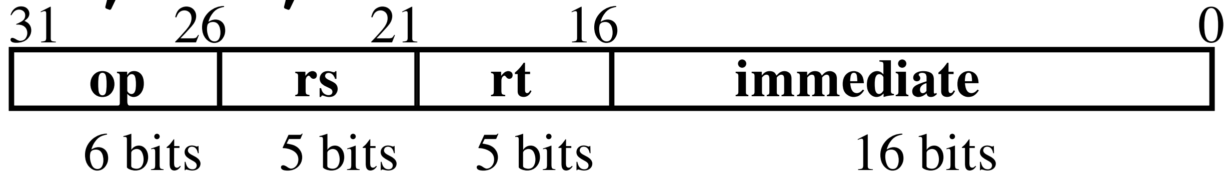
- R[rt] = Mem[R[rs] + SignExt[imm16]]
  Example: `lw rt,rs,imm16`



3e: Store Operations

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

Ex.: `sw rt, rs, imm16`

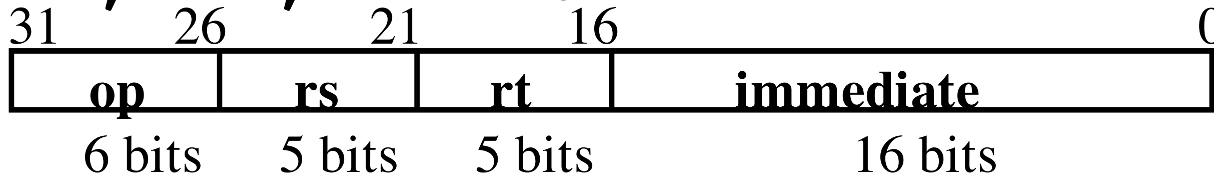


RegDst
Rd
Rt
1
0
RegWr
Rs
Rt
5
5
5
Rw
Ra
Rb
busW
32
RegFile
clk
busA
32
busB
32
0
1
Imm16
16
Extender
32
ExtOp
ALUSrc
ALUctr
ALU
32
MemWr
MemtoReg
32
WrEn
Adr
Data_In
Data Memory
clk
0
1

3e: Store Operations

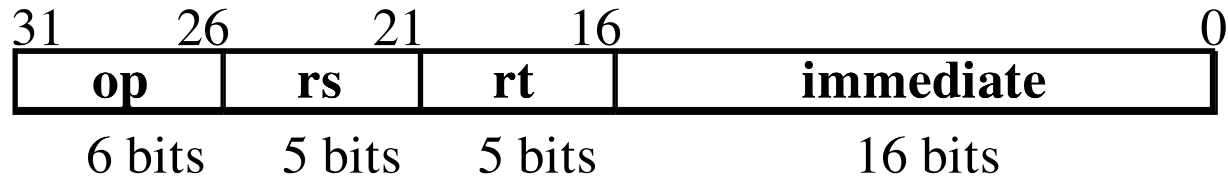
- Mem[R[rs] + SignExt[imm16]] = R[rt]

  Ex.: `sw rt, rs, imm16`



RegDst Rd Rt
1 0
RegWr
Rs Rt
5 5 5
Rw Ra Rb
busW
32
RegFile
clk
busA 32
busB
32
ALUctr
MemWr
MemtoReg
ALU
32
0
1
imm16 16
Extender
32
ExtOp
ALUSrc
32
Data In
WrEn Adr
Data Memory
clk
0
1

3f: The Branch Instruction



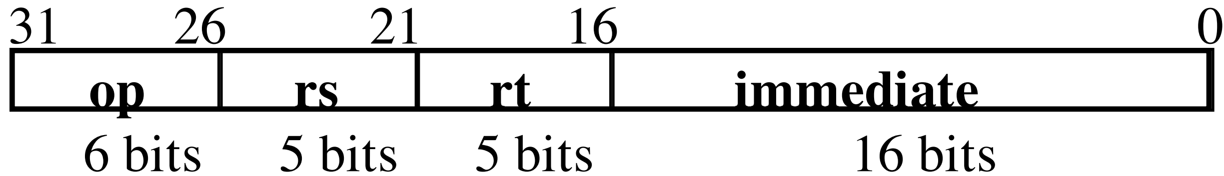

```
beq rs, rt, imm16
```

- 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)

Inst Address

nPC_sel

4

Adder

Mux

00

PC

Adder

PC Ext

clk

Imm16

Equal

RegWr

Rs

Rt

5

5

5

ALUctr

Rw

Ra

Rb

busW

busA

32

=

ALU

32

RegFile

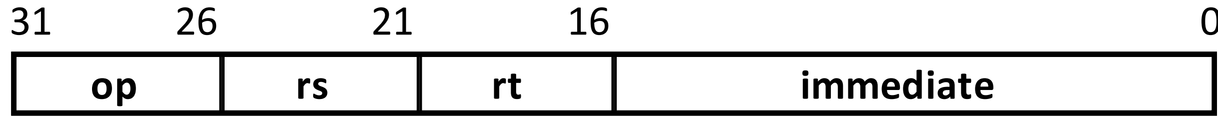
busB

32

clk

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

Instruction Fetch Unit including Branch



- if (Equal == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4

Inst
Memory
Adr
Instruction<31:0>

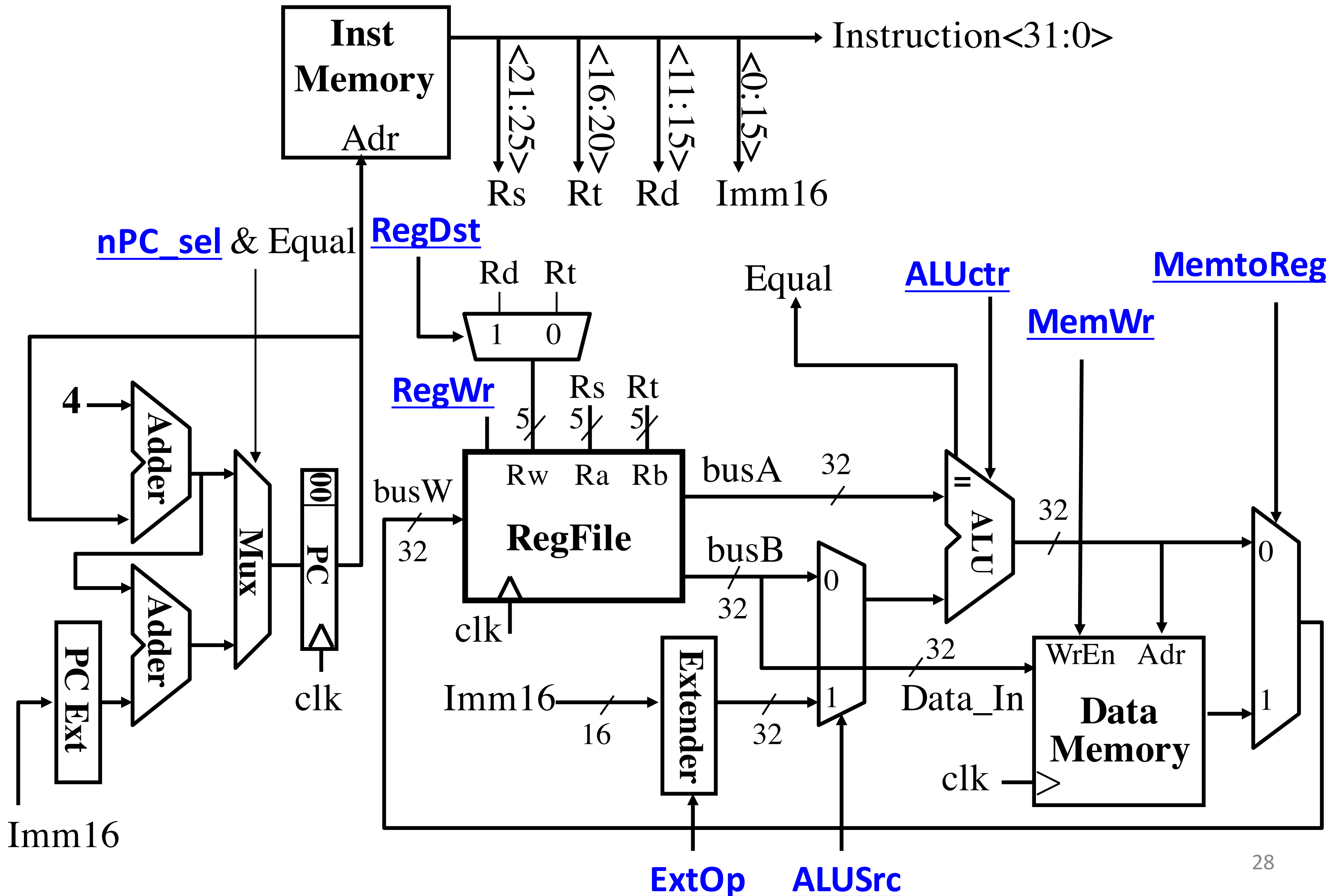
nPC_sel
Equal
MUX_ctrl
4
Adder
Adder
0
Mux
1
00
PC
clk
imm16
PC Ext

- How to encode nPC_sel?
  - Direct MUX select?
  - Branch inst. / not branch inst.
- Let' s pick 2nd option

nPC_sel	Equal	MUX_ctrl
0	x	0
1	0	0
1	1	1

Q: What logic gate?

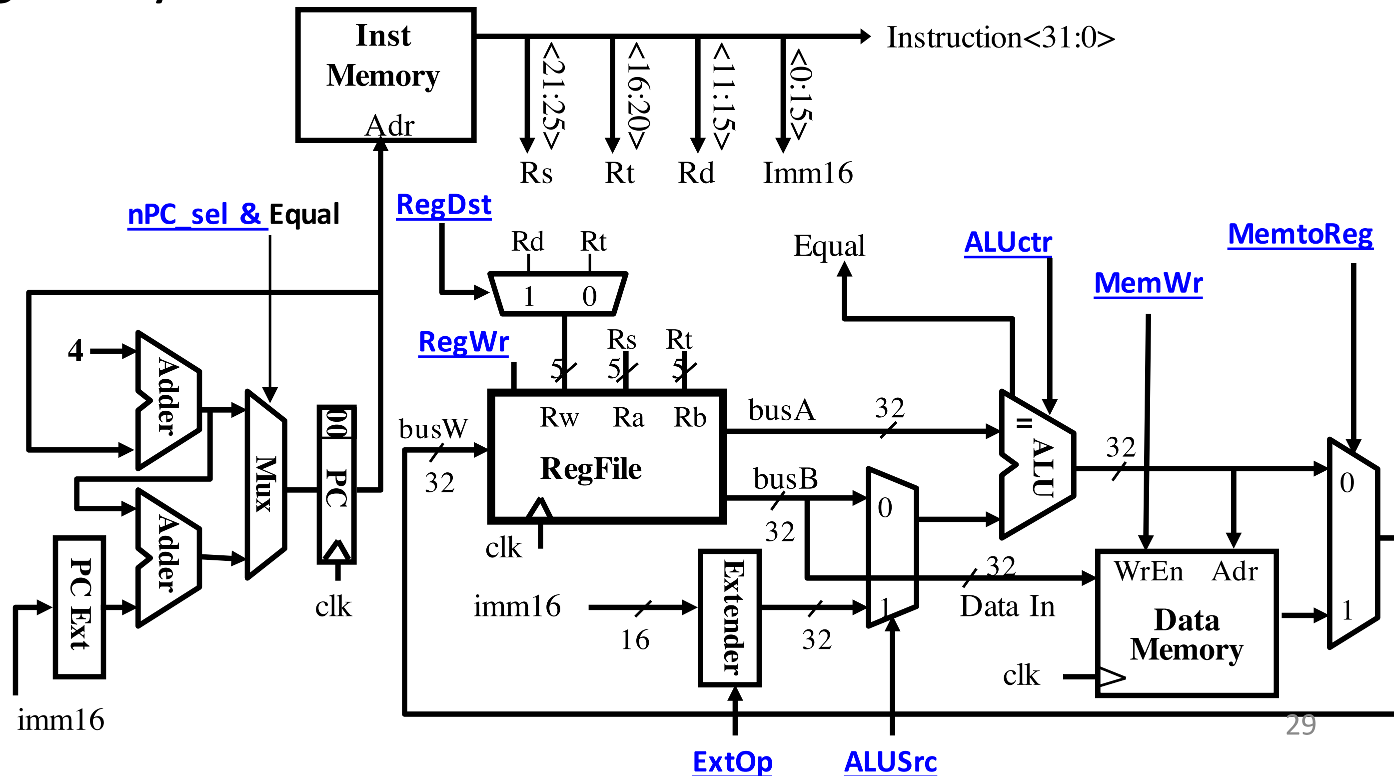
Putting it All Together:A Single Cycle Datapath



Clickers/Peer Instruction

What new instruction would need no new datapath hardware?

- A: branch if reg==immediate
- B: add two registers and branch if result zero
- C: store with auto-increment of base address:
  - sw rt, rs, offset // rs incremented by offset after store
- D: shift left logical by two bits



Administrivia

- Project 2-2 released, due 3/08 (a Tuesday!)
- Guerrilla Session:
  - Synchronous Digital Systems
  - Wed 3/02 3 - 5 PM @ 241 Cory
  - Sat 3/05 1 - 3 PM @ 521 Cory