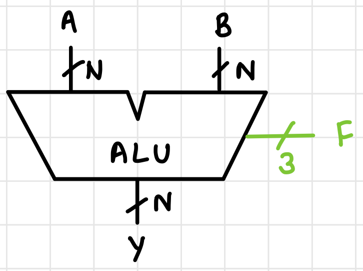
Intel and Advanced Micro Devices (AMD) both sell compatible microprocessors conforming to the x86 architecture. Intel Pentium III and Pentium 4 microprocessors were largely advertised according to clock frequency in the late 1990s and early 2000s, because Intel offered higher clock frequencies than its competitors. However, Intel's main competitor, AMD, sold Athlon microprocessors that executed programs faster than Intel's chips at the same clock frequency. Why?

- instructions / clock cycle
- frequency not the only factor; lower clock frequency but better performing
- Zen, Zen 2 - AMD microarchitectures that implement x86 architecture

MULTI-CYCLE PROCESSOR

- Processor: datapath (functional blocks), control (control signals)
  - datapath ↓ where data flows
  - control ↓ manipulates data
- Register file + ALU

ALU - Arithmatic and Logic Unit



ALU Control[2:0]

$F_{2:0}$	Function
000	A & B AND
001	A | B OR
010	A + B add
011	not used
100	A & ~B
101	A | ~B
110	A - B sub
111	SLT shift left

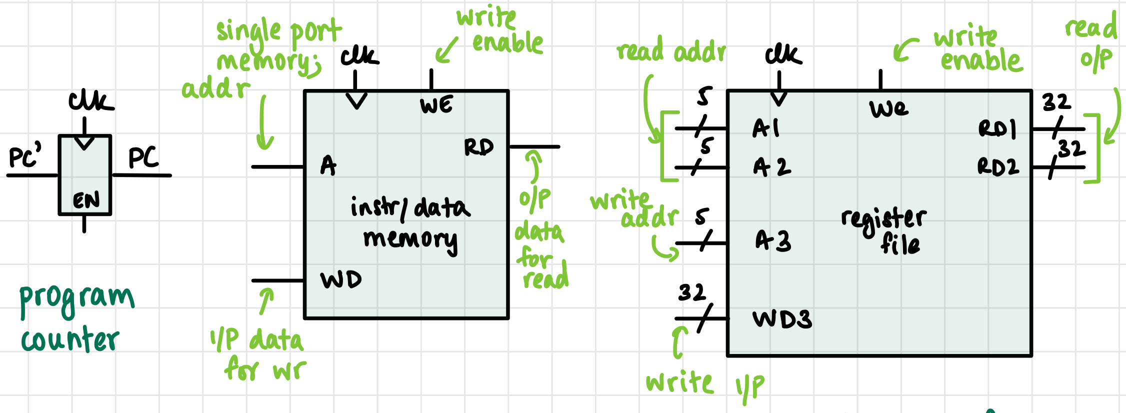
8 functions: 3 inputs

Architectural State

- PC
- Registers
- Memory

3 components

↳ outsid microprocessor ; external



recall memory arrays

recall lab – reg file

Load Word Instruction

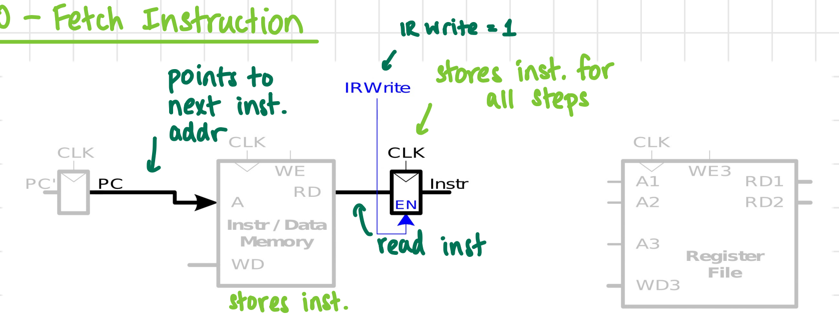
lw \$s0, 5(\$t1) → load from memory

rt (under \$s0), rs (under \$t1)

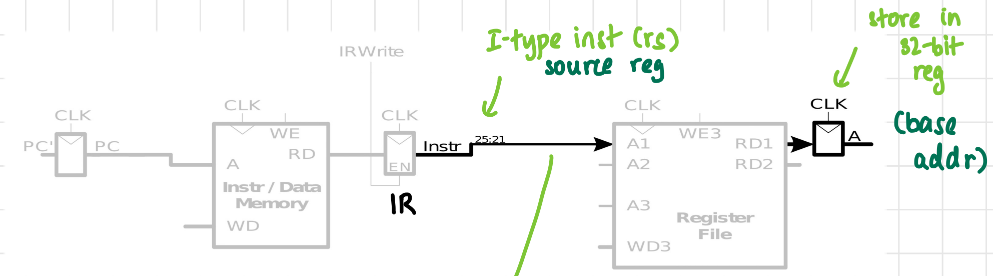
- Address calculation

  base address (\$t1) + offset (5)

Step 0 – Fetch Instruction



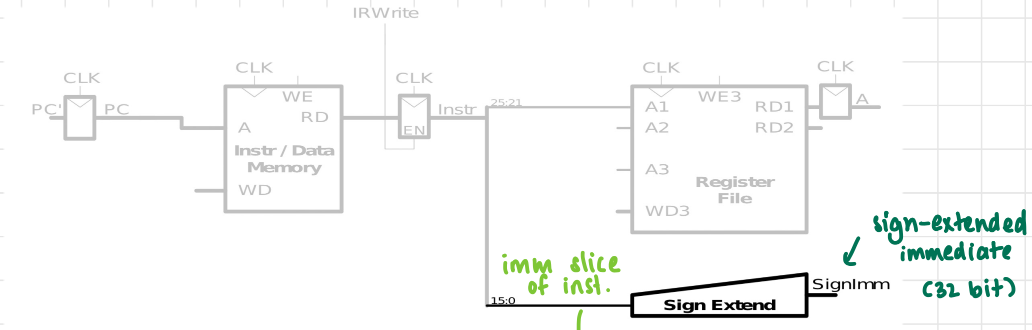
Step 1 - Read source operands from RF



Recall: I-type instruction

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
31:26	25:21	20:16	15:0

Step 2 - Read & Sign Extend the Immediate

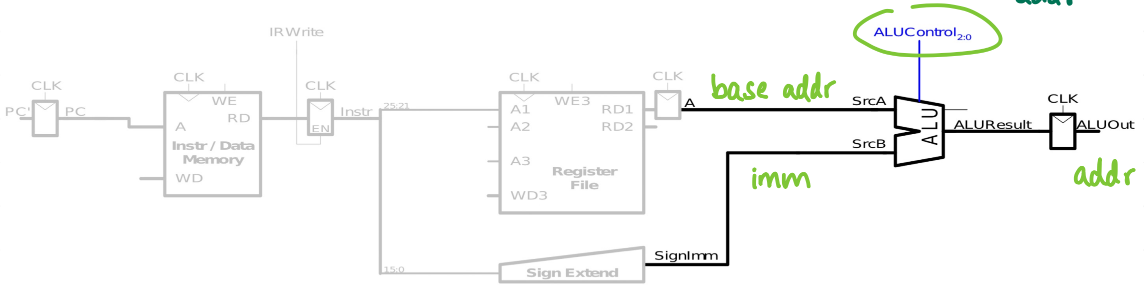


Recall: I-type instruction

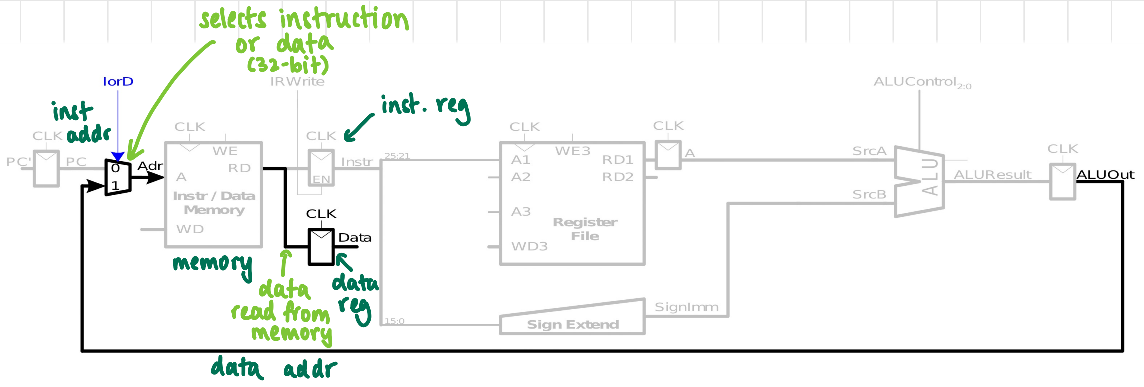
op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
31:26	25:21	20:16	15:0

Step 3 - Compute Memory Address

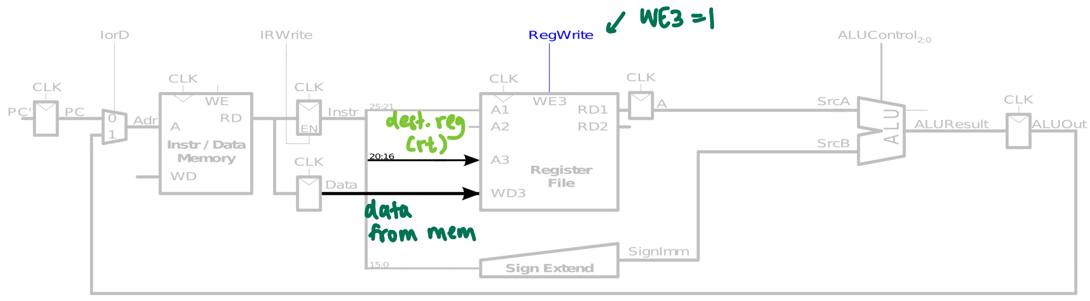
add immediate to base addr



Step 4 - Read from Memory



Step 5 - Write data back into RF

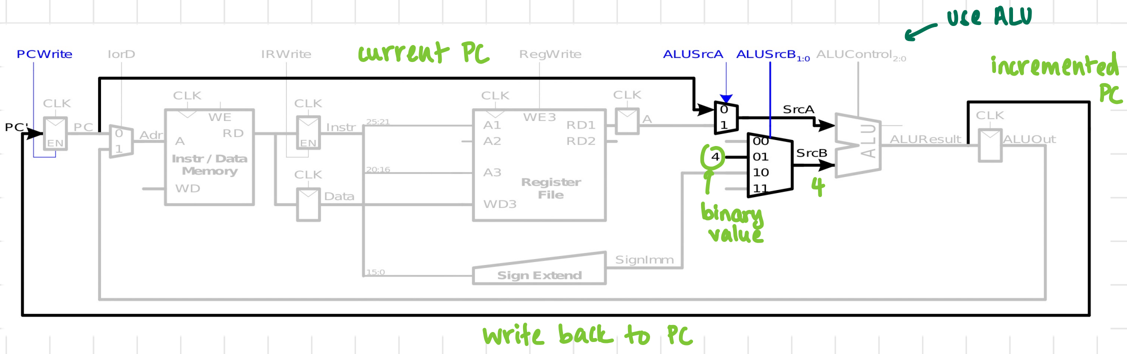


Recall: I-type instruction

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
31:26	25:21	20:16	15:0

Step 6 – Increment PC

- increment by 4 bytes (32-bit is 4 bytes, byte-addressable)



- Only one adder (inside ALU)
- In lab datapath design, two adders used (separate adder for PC)
- Would using two adders in current design help? How?
- Note that in both cases, architecture remains the same

Other Instructions

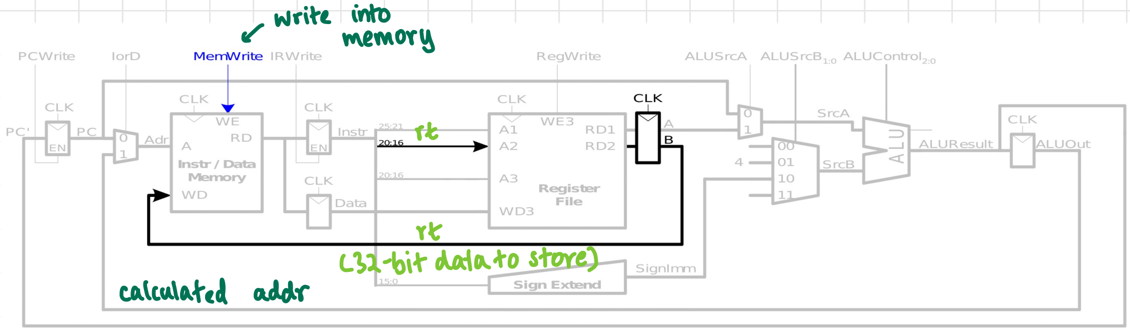
- sw
- R-type (add, sub, and, ...)
- beq

Store Word Instruction

sw $\underset{rt}{\$s0}$, $\underset{rs}{7(\$0)}$ # write the value in \$s0 to loc 7

- Address computation same as for lw
- Register contents to be written to main memory
- Steps 1, 2 and 3 same as in lw
- Step 4: Register contents to be written into computed memory address

Write to Memory



Recall: I-type instruction

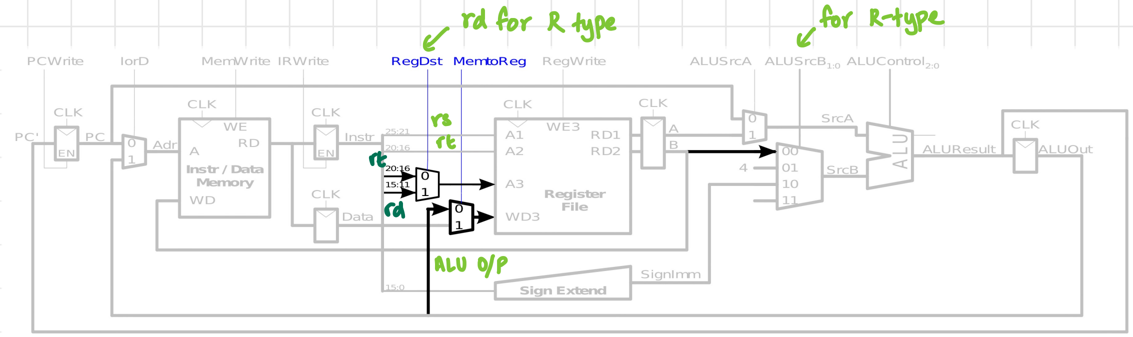
op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
31:26	25:21	20:16	15:0

R-type Instructions

- Step 1 (fetch): same as lw
- Step 2: similar to lw (no sign extending; read 2 operands)
- Step 3: write ALU output into destination register

- Read from rs and rt
- Write ALUResult to reg file
- Write to rd (instead of rt)

add $s0, $s1, $s2
rd rs rt



Recall: R-type instruction

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5bits	6 bits
31:26	25:21	20:16	15:11	10:6	5:0

beq Instruction

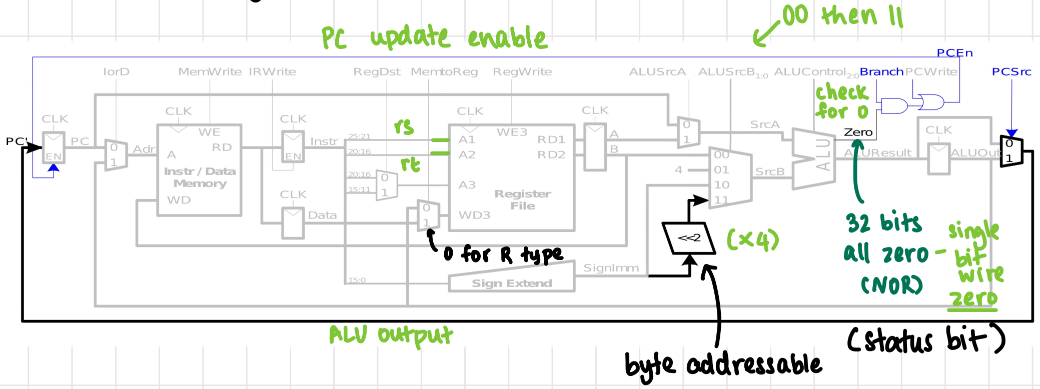
- Step 1: fetch
- Step 2: compare register contents
- Step 3: change PC contents (if registers equal)

beq \$s0, \$s1 , loop (PC relative)

rs == rt?

BTA = (sign-extended immediate << 2) + (PC+4)

offset value x4 (<<2) next PC inst



Recall: I-type instruction

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits
31:26	25:21	20:16	15:0