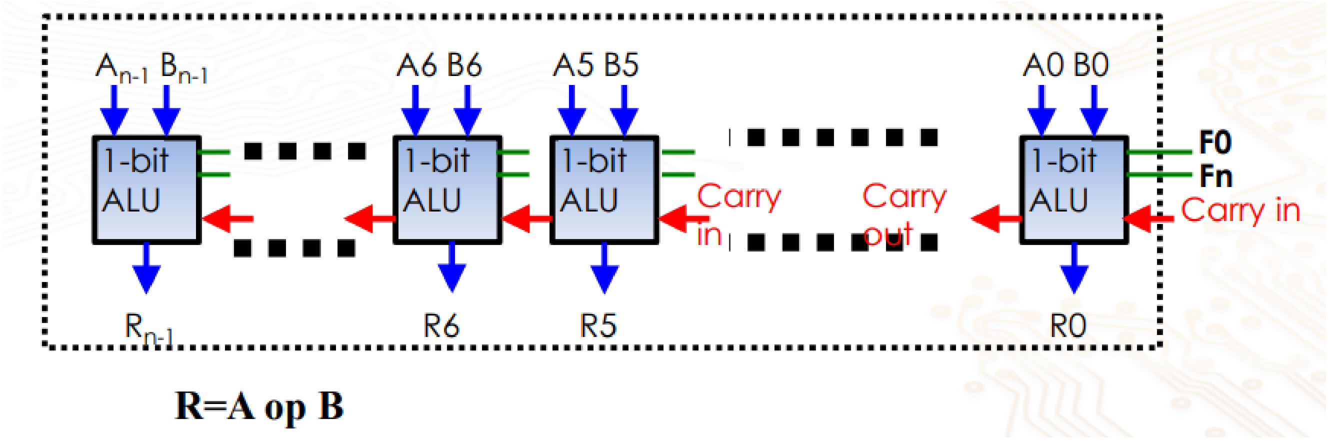
M3 Data Path and Control Design

1. Datapath components

1.1 Arithmetic and Logic Unit (ALU)

- CPU's data processing / execution unit
  - 3 inputs (A, B, function select)
  - 1 output (out)
- Implements **fixed point** operations
  - floating point and complex number functions are performed by arithmetic co-processors
- Functions
  - Arithmetic
    - Add, Sub, Mul, Div
  - Logical
    - AND, OR, XOR, NOT
  - Data manipulation
    - Arithmetic / Logical shifts, incrementing, decrementing of operands (A, B)
- n-bit ALU is made by placing n 1-bit ALU slices in parallel, function select for each ALU is driven by the instruction being executed



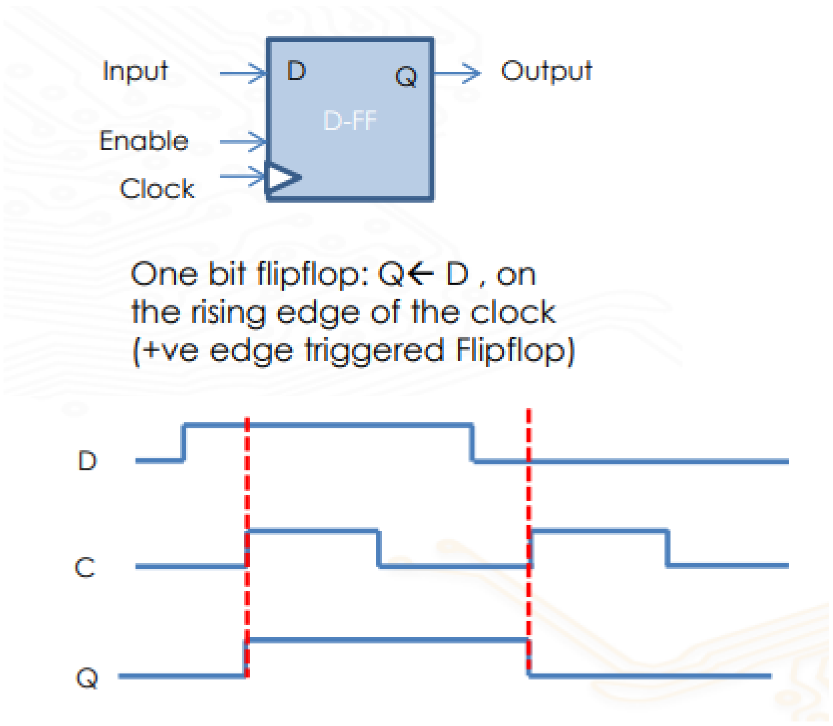
1.2 Register File

- Collection of registers, can be read / written by specifying address, from the temporary storage in CPU

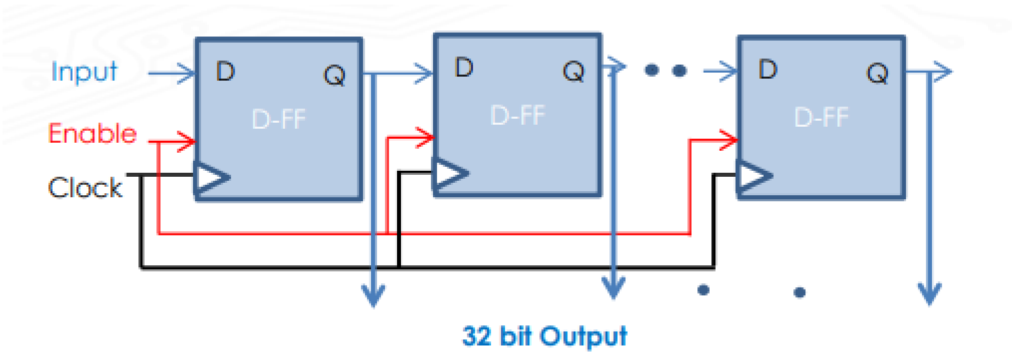
Building Blocks

- **Multiplexer x 2**

    - 32:1 MUX (32 inputs, 1 output) in MIPS register file
- **Decoder**
    - 5 : 32 decoder (5 bit input, 32 outputs) in register file
    - Only one output of a decoder is high at any given time depending on input
        - Can be used with RegWrite signal (of register file) to write to register file
- **Register**


Input
Enable
Clock
D
Q
D-FF
Output
One bit flipflop: Q← D , on the rising edge of the clock (+ve edge triggered Flipflop)
D
C
Q

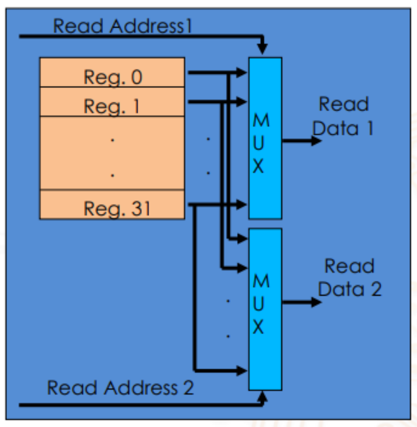
    - Made up of an array of D-flipflops to hold multi-bit data such as bytes / words



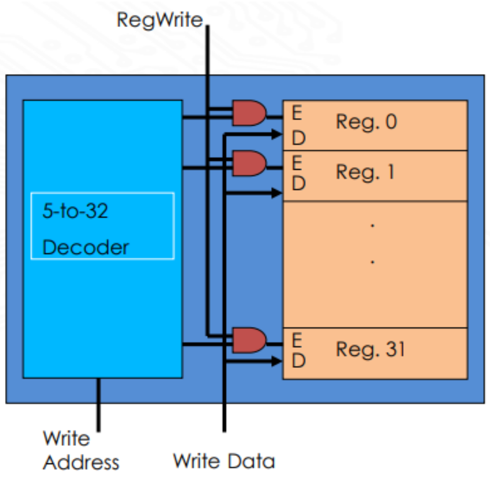
    - Used to build datapath

Read / Write Ports of MIPS Register File

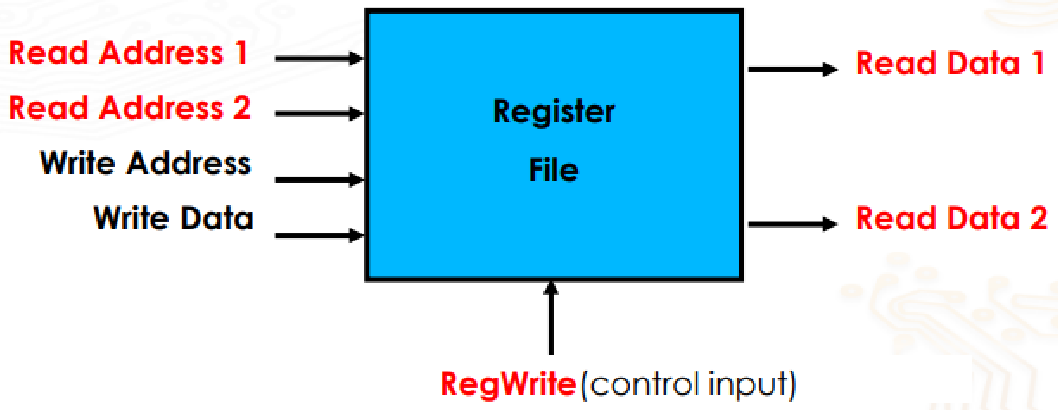
- Register File consists of a set of registers that can be read and written by supplying a target register number
    - Implemented with a **multiplexer for each read** and a **decoder to write** and array of registers built from D-flipflops



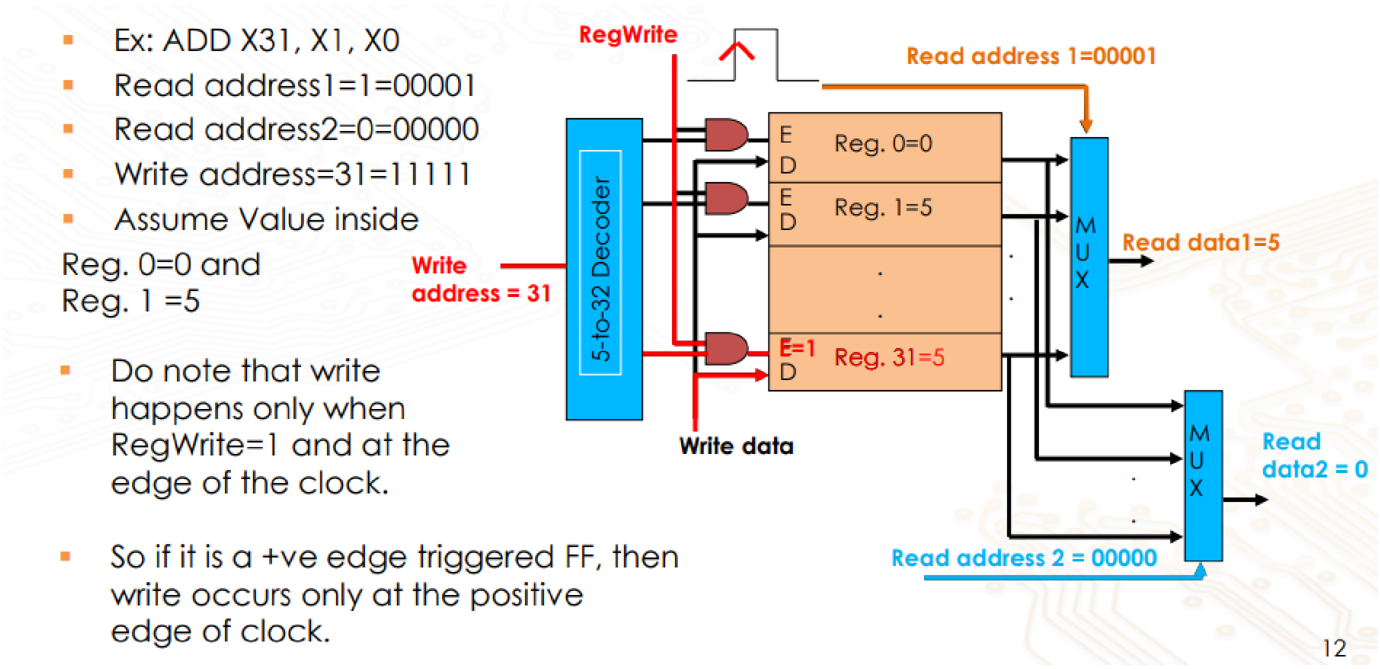
Read Ports for a Reg. File



- Combining both read and write



Working of Register File



- Ex: ADD X31, X1, X0
- Read address1=1=00001
- Read address2=0=00000
- Write address=31=11111
- Assume Value inside

Reg. 0=0 and
Reg. 1 =5

- Do note that write happens only when RegWrite=1 and at the edge of the clock.
- So if it is a +ve edge triggered FF, then write occurs only at the positive edge of clock.

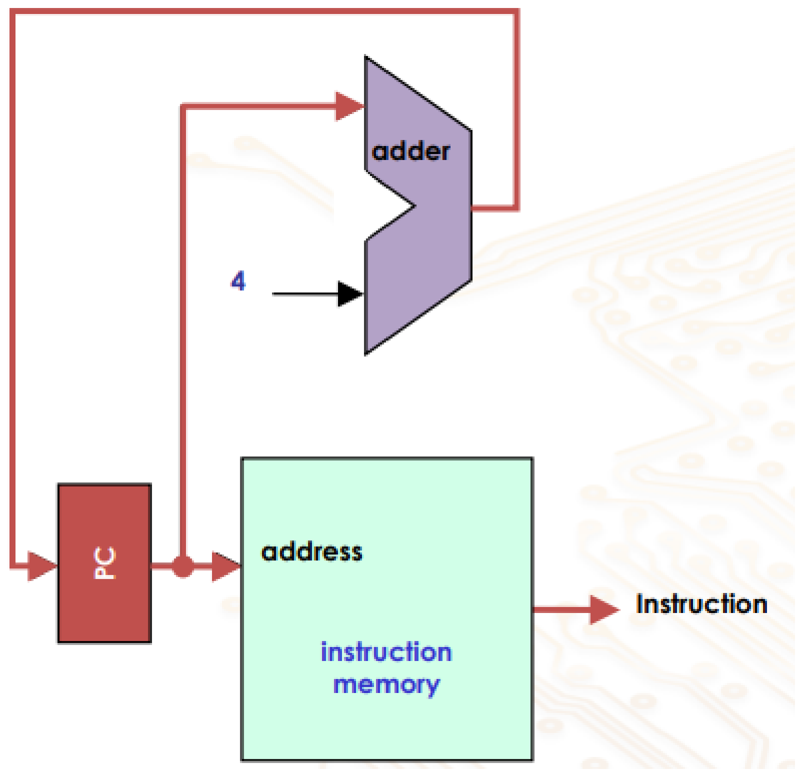
- data1 and data2 goes to ALU, write data comes from ALU
- Write address comes from destination register (X31 this case)
- Decoder will enable register together with RegWrite (1 AND 1), data will then be written into register
    - Register can only be written to when it is enabled
- **When the same register is read and written during the same clock cycle**

    - Register read happens combinational (can be read at anytime)
    - Register valid during time it is read
        - Value returned is the value written in the previous clock cycle
    - Write of register file happens on clock edge (sequentially)
    - **Additional logic is needed to return (by reading) the value currently being written**

2. Instructions

2.1 Instruction Fetch

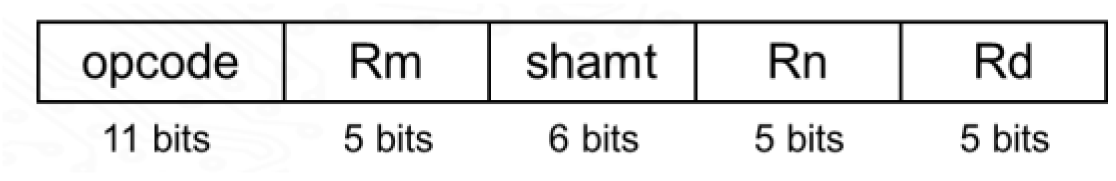
- Instruction Memory stores instructions of programs and supply instructions when given an address
    - Only needs read access, no read control signal needed
    - Treated as combinational logic
- PC (register) holds address of current instruction (unlearn 1106) and is written at the end of every clock cycle (next rising edge, PC is enabled at RegWrite)
    1. Fetch current instruction from address pointed to by PC
    2. Increment PC by 4 to point to next instruction 4 bytes later (each instruction is 4 bytes)



2.2 Datapaths / transfer

It is important to know the datapaths for each type of instruction

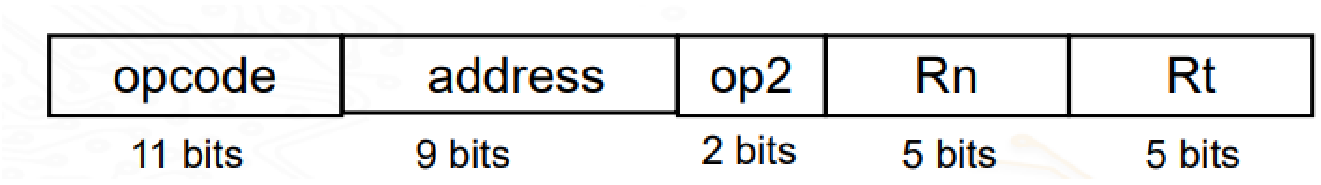
R-type



opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

- **Data Transfer**
    - All data values located in registers, addressing mode is Register addressing
    - Instruction read from instruction memory (instruction fetch IF)
    - Source registers Rn Rm read from register file and fed to ALU (instruction decode ID)
    - ALU performs operation specified by opcode (execute EXE)
    - Result stored in destination register Rd (write back WB)
- **Datapath (refer to D-type)**

D-type



opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits

- **Data Transfer**
    - `LDUR Rt, [Rn, #address]`
        - Source register Rn read and fed to ALU with `address` after sign extension to 64 bits
        - ALU calculates memory address of word to be loaded (EXE)
        - Content (word size) read from memory address (MEM)
        - Content written back to register Rt (WB)
    - `STUR Rt, [Rn, #address]`
        - Source register Rn read, fed to ALU with `address` after sign extension to 64 bits
        - ALU calculates memory address of data to be stored (EXE)
        - Result stored at memory address calculated in EXE
- **Datapath**

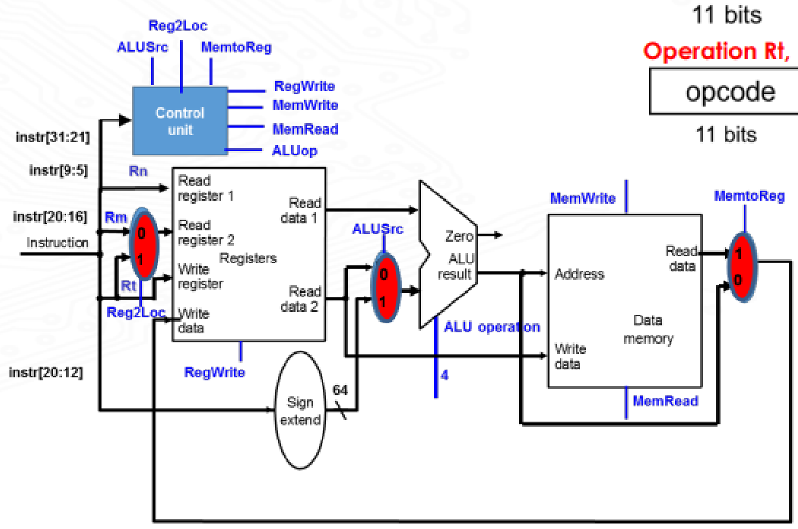
How can we combine the data path for R and D type instructions?

Operation Rd, Rn, Rm

opcode	Rm	shamt	Rn	Rd
11 bits	5 bits	6 bits	5 bits	5 bits

Operation Rt, [Rn, address]

opcode	address	op2	Rn	Rt
11 bits	9 bits	2 bits	5 bits	5 bits



"Rn" -Source for R and D type
"Rd" -Destination for R-type
"Rt" -destination/source for D-type

Extra mux needed

"Reg2Loc" -Selects between "Rt" and "Rm" as the source register address
"ALUSrc" -Selects between "read data2" and "address" as the source to ALU
"MemtoReg"-select the result from memory or from ALU

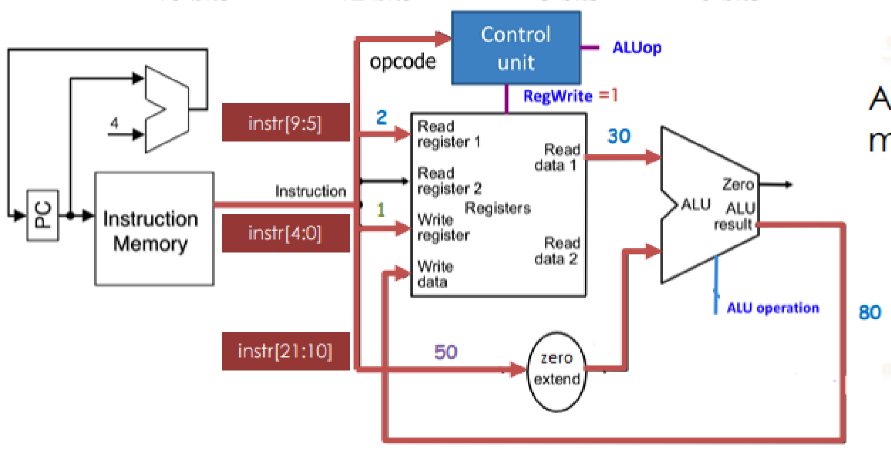
I-type

opcode	immediate	Rn	Rd
10 bits	12 bits	5 bits	5 bits

- **Datapath**

580	50	2	1
Opcode 10 bits	immediate 12 bits	Rn 5 bits	Rd 5 bits

Register	Value
X2	30

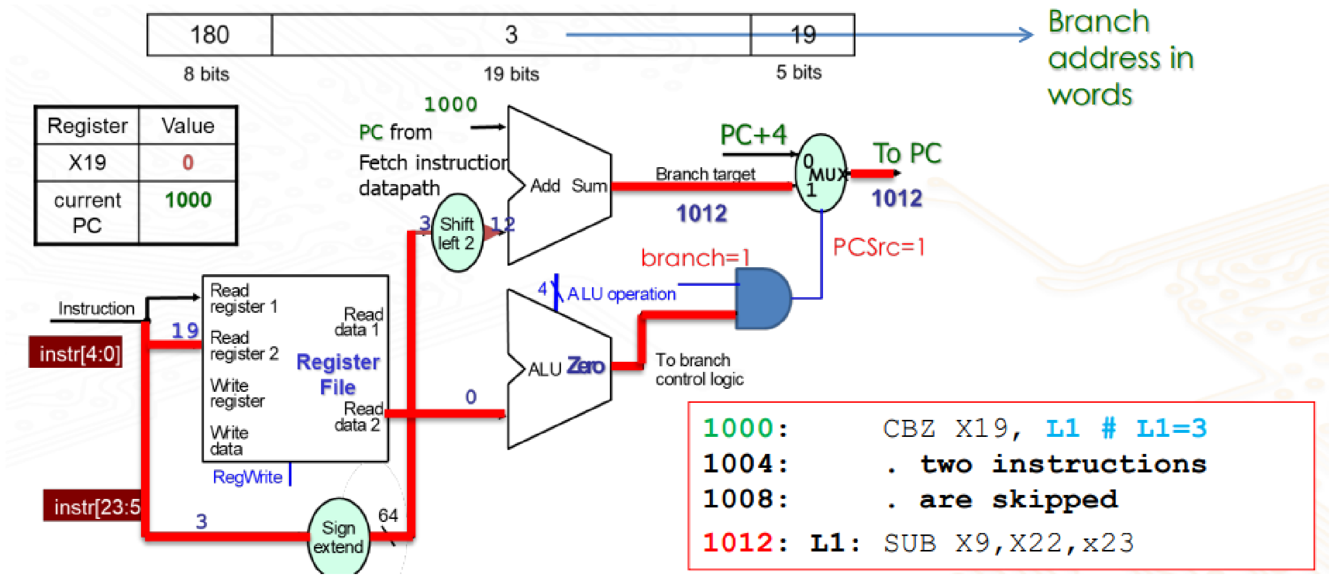


ADDI X1, X2, #50
meaning([X1] ← [X2]+ 50])

CB-type

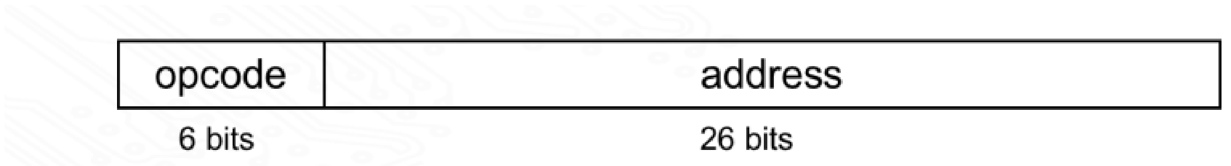
opcode	address	Rt
8 bits	19 bits	5 bits

- **Datapath**



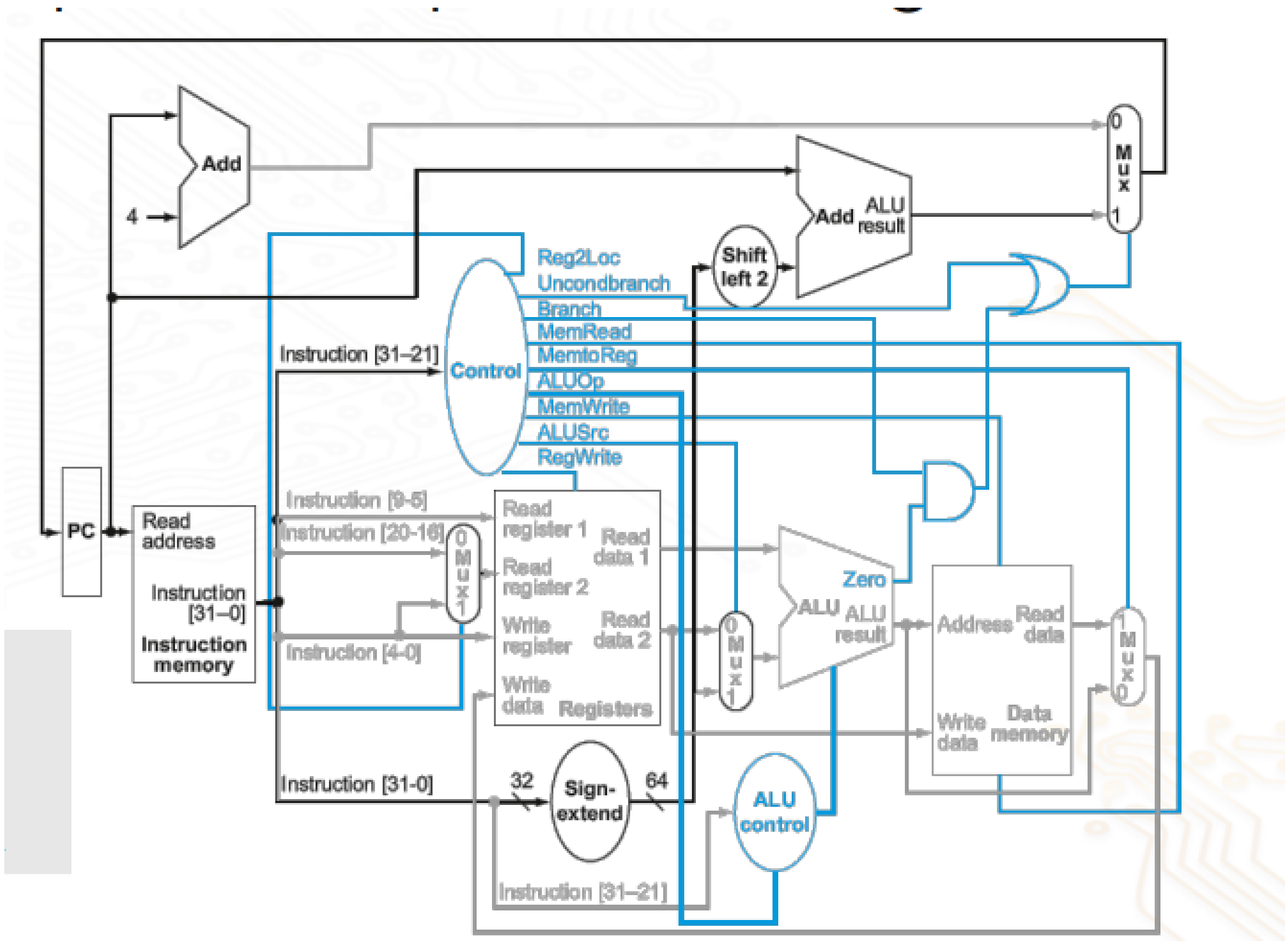
```
1000:      CBZ X19, L1 # L1=3
1004:      . two instructions
1008:      . are skipped
1012: L1:  SUB X9,X22,x23
```

B-type



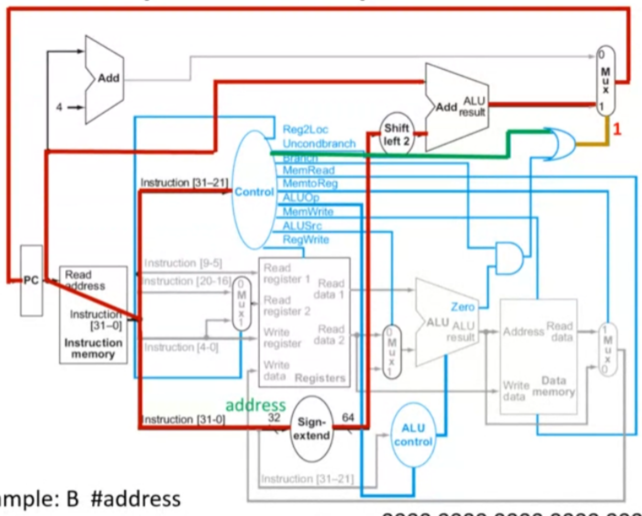
- Need an extra control signal decoded from opcode
- `address` is manipulated before putting into PC
  - All instructions are 32 bits, this means that `address` is manipulated into 32 bits before passing into the PC
  - 4 most significant bits are truncated, hardware reuses 4MSB from the previous instruction because jump is limited by the number of addresses available and cannot extend into the 4MSBs
  - 2 bits are padded to the right, increasing number of bits for address to 28
  - This means branch can move ± 128bits from PC
- https://www.youtube.com/watch?v=oETOwVBzu1s
- **all branching needs to shift left twice since memory is byte addressable (4bits)**

Complete datapath combining all types



- Consider a modified version of LEGv8 processor with 32 bit of data and instruction bus. Find the maximum address of the instruction memory to which the control of execution of LEGv8 code could be moved forward by the unconditional branch instruction of the form "B offset"

Complete Datapath



- 26 bits goes into sign extend
- if value is positive, then it remains positive
- if value is negative then it remains negative
- This value is multiplied by 4 (left shifted by 2).
- 26 bits will be left shifted by two making it to be effectively 28 bits
- Do note that multiply by 4 (3'b100), is appending two zeros towards the end.

That is why we get 28 bits, as it is sign extended 28th bit will be sign bit and hence the range is within +-(128 MB)

max Positive number before sign extension (26 bits)

(max positive number)

01 1111 1111 1111 1111 1111 1111

max neg number 10 0000 0000 0000 0000 0000 0000

2 0 0 0 0 0 0

Positive number after sign extension (64 bits)- zero extension

6

example: B #address

opcode	address
6 bits	26 bits

0000 0000 0000 0000 0000 0000 0000 0000 0000 0001 1111 1111 1111 1111 1111 1111

0 0 0 0 0 0 0 0 0 1 F F F F F F

After multiply by 4

0000 0000 0000 0000 0000 0000 0000 0000 0000 0111 1111 1111 1111 1111 1111 1100

$0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; 0 \; \underbrace{0 \; 7 \; F \; F \; F \; F \; F \; C}_{\text{32 bit data}}$

(21 Feb 2022 review)