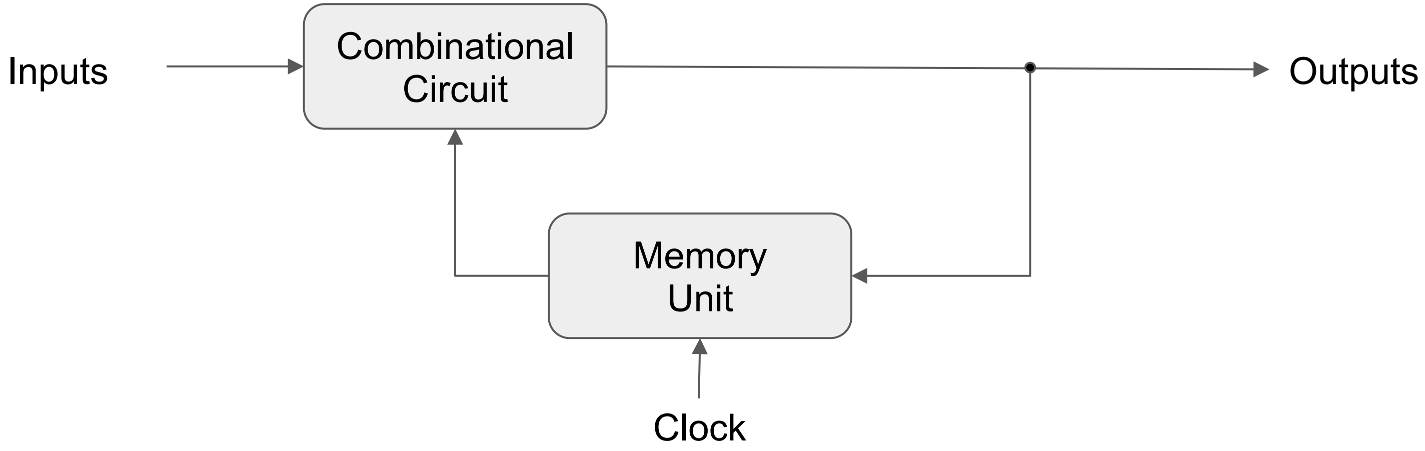
Lecture:

- Last Time:
    - Control Lines and the CPU
        - On-Off Switch
        - Decoder to select which register.
        - Mux to select ALU Output
        - CPU Overview: Data with some control lines.
        - High-level memory schematic
    - Questions

- Today:
    - Sequential Circuits: clocks, latches, and flip/flops:
    - CPU Control: PC/IR
    - MIPS pipeline architecture

Sequential Circuit

- Sequential circuit infuses a memory component
- Memory serves as both input and output

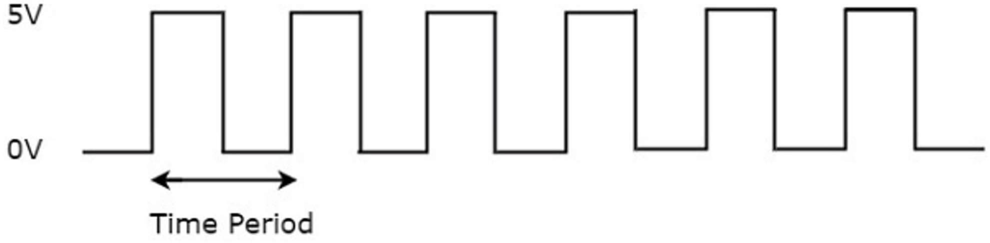


- A clock is used to trigger the update of the memory unit

Clock: an electronic device

- Generates a periodic signal
- Forms a square wave
- Frequence = 1 / (Time Period)
- Used as a control signal to active parts of a circuit

- Types of Triggers:*
  - Level Trigger
  - Edge Trigger
- Level of Triggers:
  - Positive Level (Hi)
  - Negative Level (Lo)

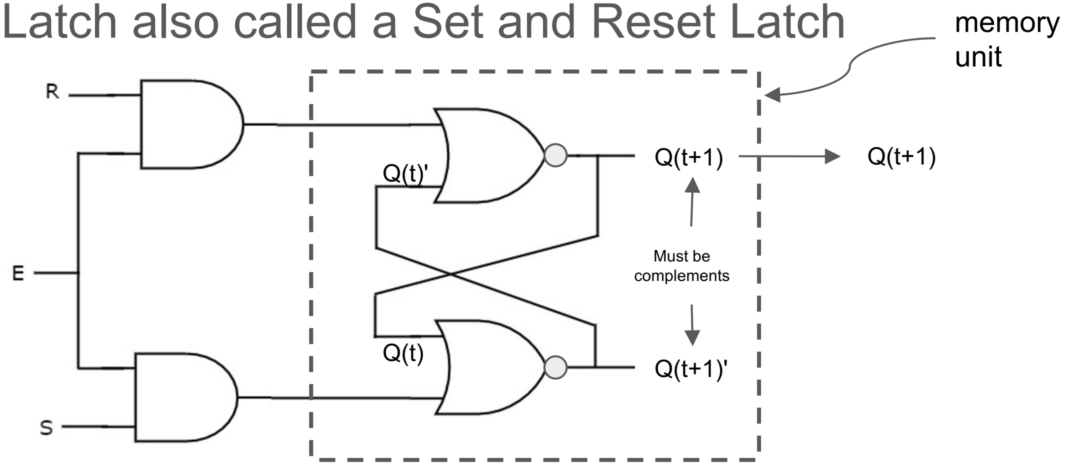


* Recall you can active some GUI elements on a mouse click, mouse press, **OR** a mouse release

Q(t)

SR Latch

- SR Latch also called a Set and Reset Latch



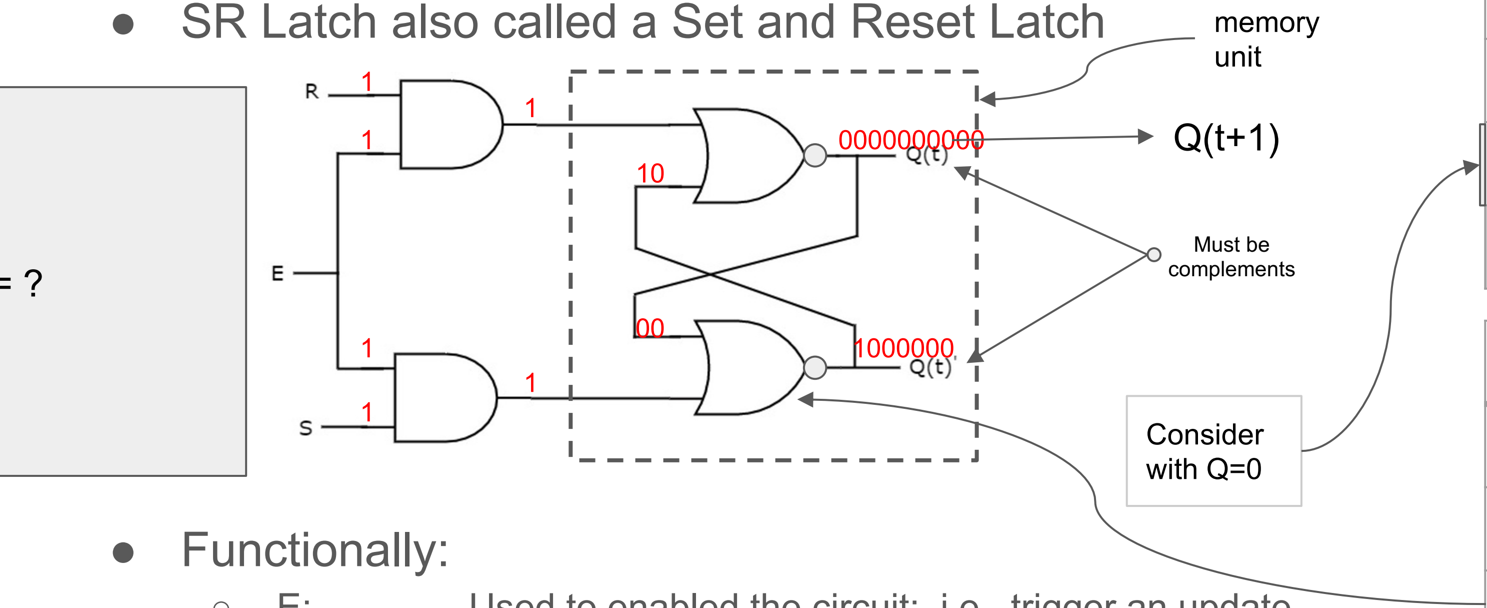
- Functionally:
  - E: Used to enabled the circuit: i.e., trigger an update
  - R: Used to clear (reset) the memory unit
  - S: Used to set the memory unit

S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	-

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

SR Latch: Set at (1,0)

- SR Latch also called a Set and Reset Latch



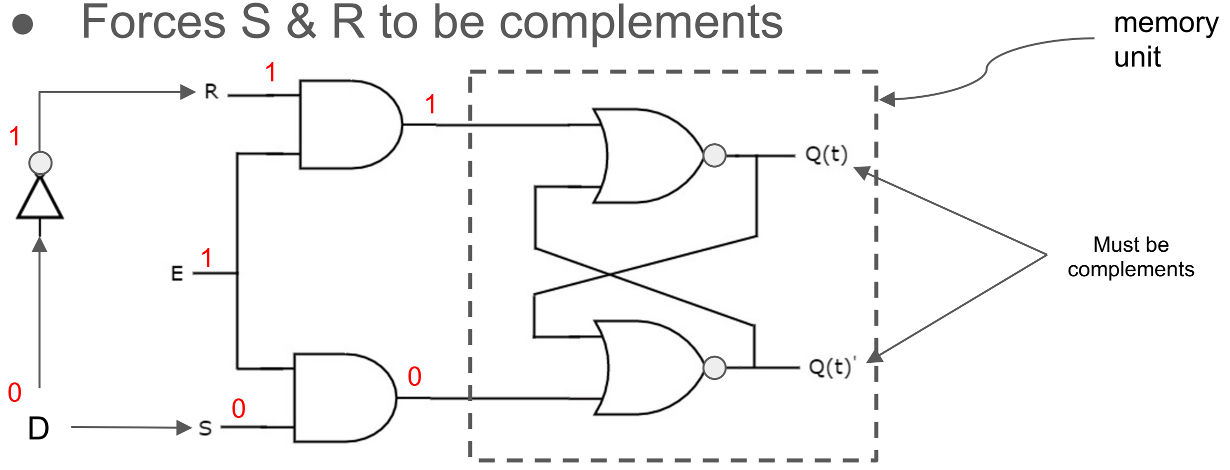
S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	-

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

- Functionally:
    - E: Used to enabled the circuit: i.e., trigger an update
    - R: Used to clear (reset) the memory unit
    - S: Used to set the memory unit

D Latch: Data Latch

- Forces S & R to be complements



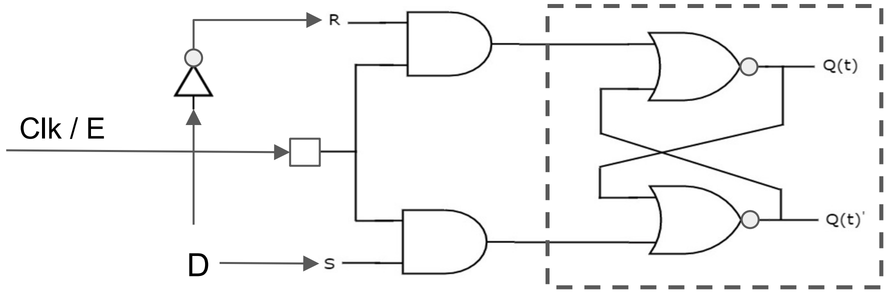
- Functionally:
  - E: Used to trigger the memory unit, i.e., perform a write
  - D': Used to clear (reset) the memory unit
  - D: Used to set the memory unit

E	D	Q(t+1)
0	0	Q(t)
0	1	Q(t)
1	0 (reset)	0
1	1 (set)	1

A	B	NOR
0	0	1
0	1	0
1	0	0
1	1	0

Latches and Flip-Flops

Logically the same!



- Latches:
  - RS Latch
  - D Latch
  - JK Latch
  - T Latch

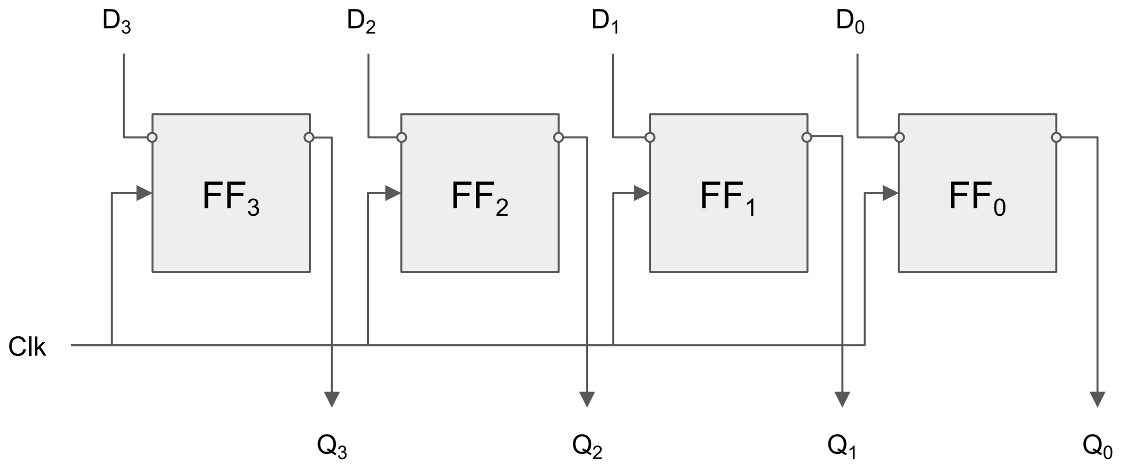
- Flip-Flop:
  - RS Flip-Flop
  - D Flip-Flop
  - JK Flip-Flop
  - T Flip-Flop

Differences:

Latches	Flip-flops
Different underlying technology is used	
Uses a enable signal	Uses a clock signal
Level triggered	Edge triggered
Consumes less power	Consumes more power
Faster	Slower
Used for transitioning (Consider a lock, e.g., at the Panama Canal.)	Used as storage

4-bit Register

- D Flip-Flop



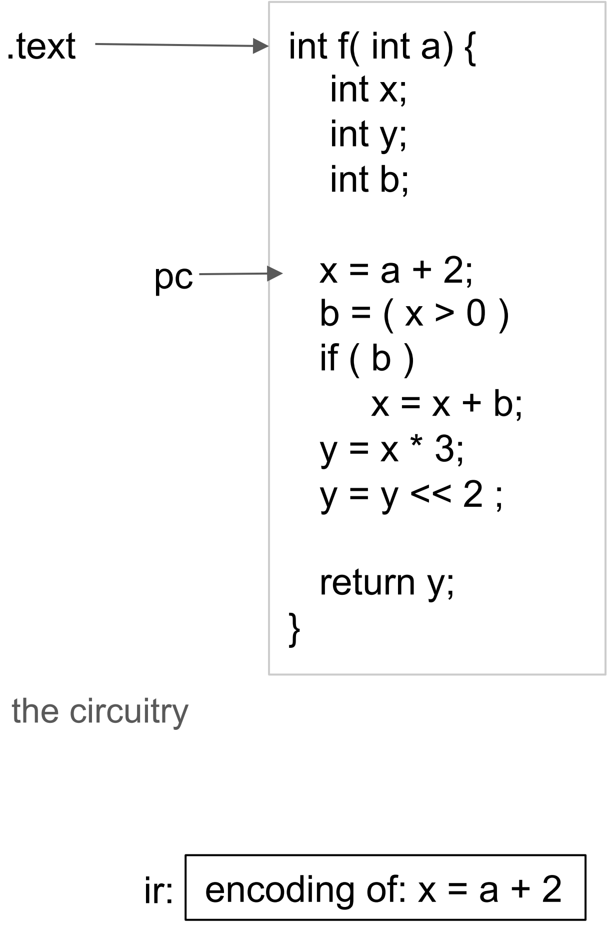
What do we have now?

1. Basic understanding of circuitry and its relationship to Boolean algebra.
2. Knowledge of "state" and how it is used to build registers and memory units
3. Recognize how we can build components to build a CPU
   - ALU, registers, decoders, multiplexers, etc., as well as main memory.
4. Almost ready for the abstract all this away in favor of an ISA
   Instruction Set Architecture

But first, let's put all the pieces together to create the MIPS microarchitecture!

MIPS: CPU Control

- The CPU executes specific circuitry in stepwise fashion
  - Fetch, Decode, Execute, Mem Access, WriteBack
  - Unnamed Latches are used between each phase
- PC: Program Counter
  - Holds the address of the NEXT instruction to be executed
  - The PC is increment by 1 instruction, i.e., 4 bytes, every cycle
  - Each instruction requires 4 bytes to encode
- IR: Instruction Register
  - Stores the current instruction to be executed
  - Each instruction is decoded (electronically) to trigger operations in the circuitry
  - Three types of encodings: R, I, J
    - Operations involving the **R**egisters and ALU
    - Operations involving **I**mmediate values
    - Operations involving **J**umps and Branches



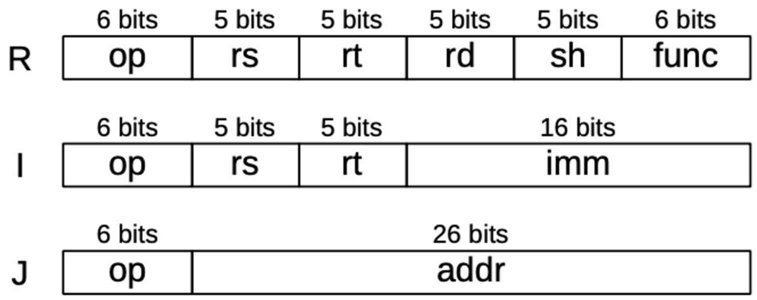
```
int f( int a) {
   int x;
   int y;
   int b;

  x = a + 2;
  b = ( x > 0 )
  if ( b )
      x = x + b;
  y = x * 3;
  y = y << 2 ;

  return y;
}
```

MIPS Instruction Set

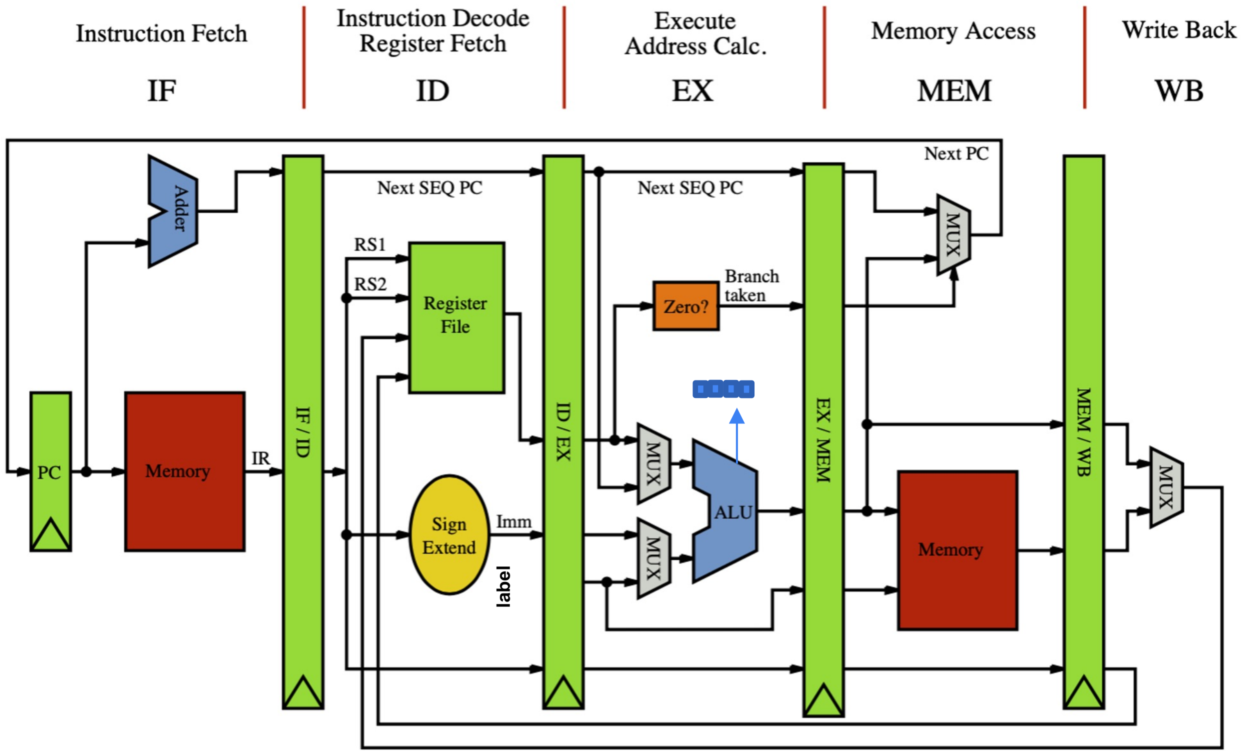
Encoding: Instruction Register



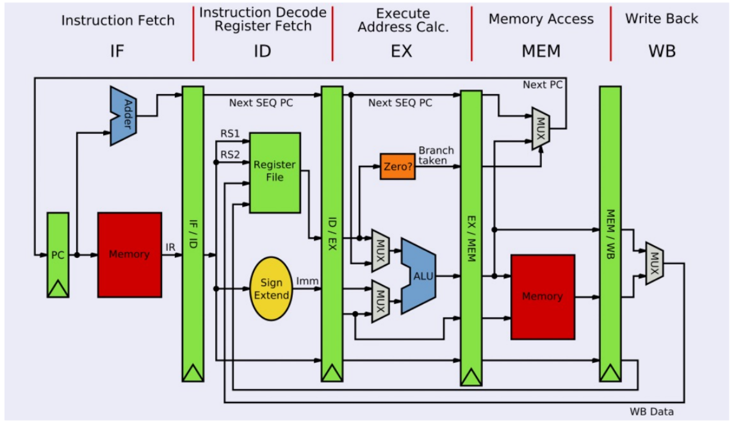
- Jump and Branch examples:
  - jal *address* ⇔ function call
  - jr rs ⇔ function return
  - beq rs, rt, *imm* ⇔ if (rs == rt) goto *imm*
- ALU examples:
  - add rd, rs, rt ⇔ rd = rs + rt
  - ori rd, rs, *imm* ⇔ rd = rs | SignExt(*imm)*
- Memory examples:
  - la rs, *label* ⇔ → \$rs = &A
  - lb rt, imm(rs) ⇔ rt = SignExt(M_1 [rs + *imm*]) → \$rt = (char) A[0]
  - lwu rt, imm(rs) ⇔ rt = M_4 [rs + *imm*] → \$rt = (unsigned word) (A[16])
  - sh rt, imm(rs) ⇔ M_2 [rs + *imm*] = rt → A[32] = (half) \$rt

MIPS Pipeline Architecture

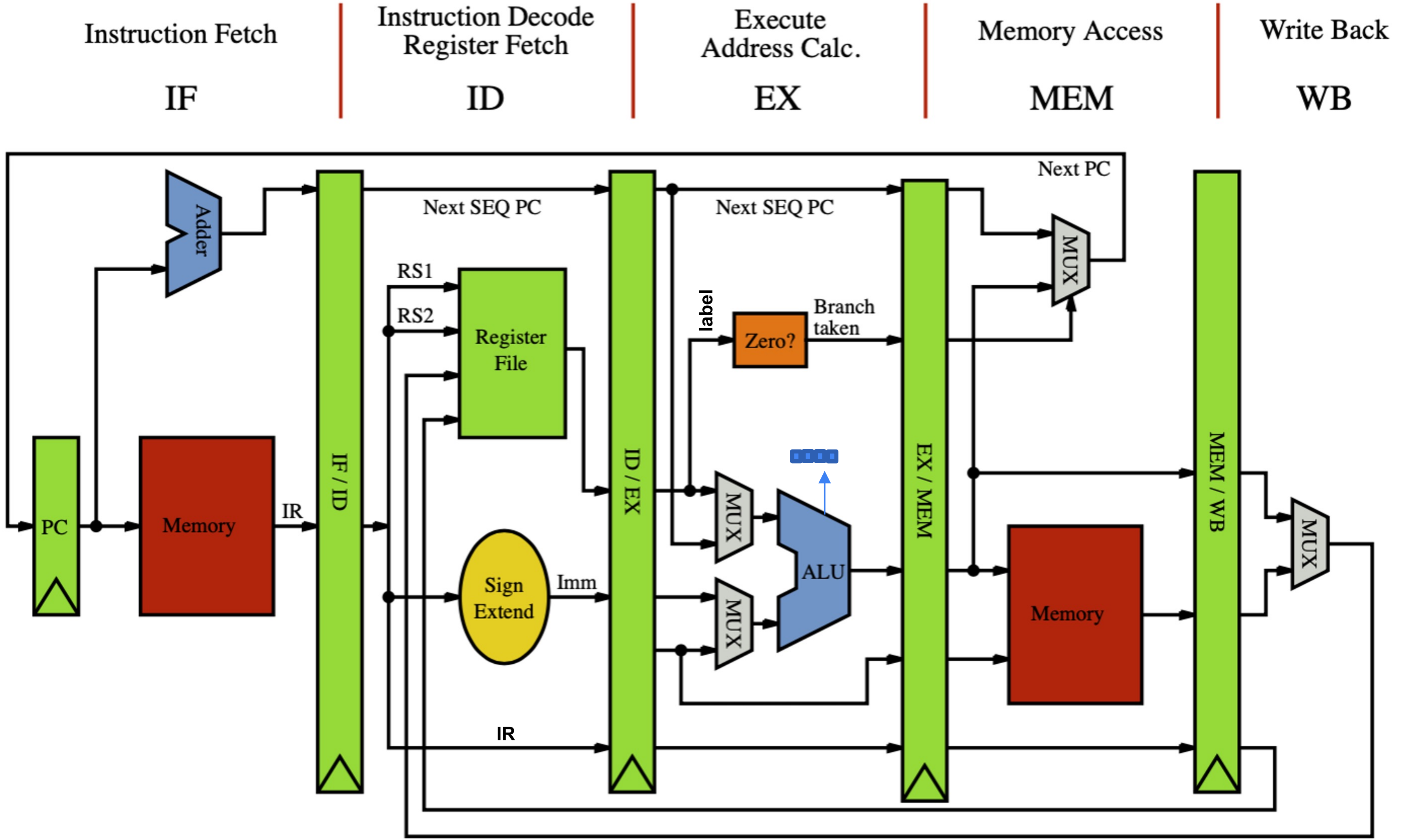
- Fives Stages →
- Major Sections:
  - PC update
  - ALU
  - Memory
- PC update
  - next instruction
  - branch
  - (PC + 4) / ALU mux
- ALU
  - imm / register mux
  - register / PC mux
- Memory
  - Harvard Model
  - ALU / memory mux



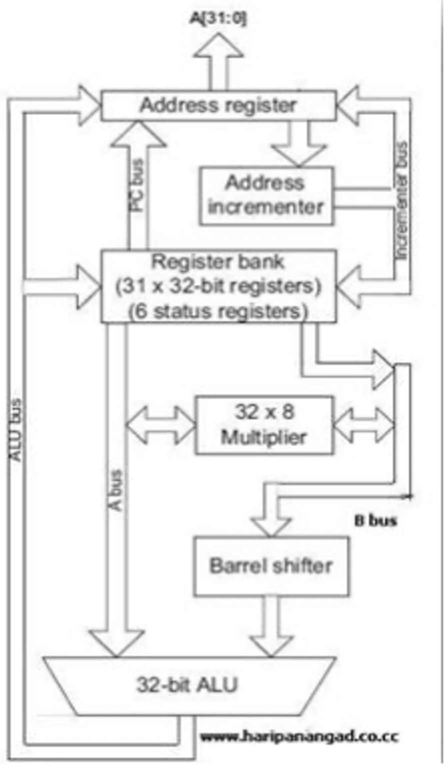
MIPS Pipeline Execution

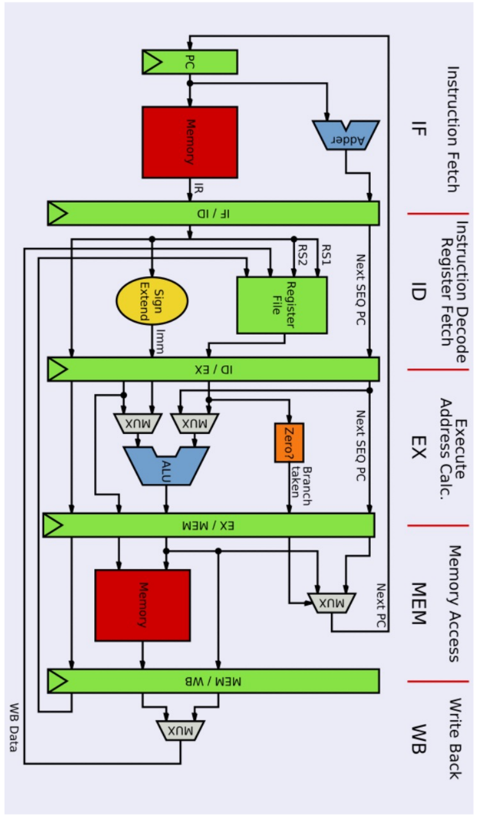


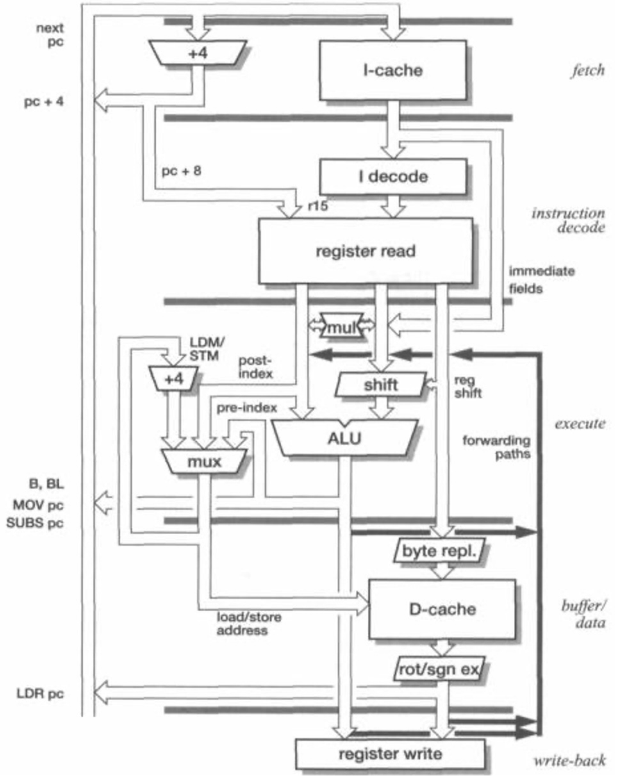
Instruction	Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
#1	Fetch	Decode	Execute	Mem	WB	Fetch #6		
#2		Fetch	Decode	Execute	Mem	WB		
#3			Fetch	Decode	Execute	Mem	WB	
#4				Fetch	Decode	Execute	Mem	WB
#5					Fetch	Decode	Execute	Mem


Instruction Fetch
IF
Instruction Decode
Register Fetch
ID
Execute
Address Calc.
EX
Memory Access
MEM
Write Back
WB
Next PC
Adder
Next SEQ PC
Next SEQ PC
RS1
RS2
Register
File
label
Zero?
Branch
taken
MUX
PC
Memory
IR
IF / ID
ID / EX
MUX
ALU
MUX
EX / MEM
Memory
MEM / WB
MUX
Sign
Extend
Imm
IR

Physical Architectures: Examples


A[31:0]
Address register
PC bus
Address incrementer
Incrementer bus
Register bank
(31 x 32-bit registers)
(6 status registers)
ALU bus
32 x 8 Multiplier
A bus
B bus
Barrel shifter
32-bit ALU
www.haripanangad.co.cc


PC
Memory
Adder
IR
IF / ID
RS1
RS2
Sign Extend
Register File
Next SEQ PC
Imm
ID / EX
MUX
MUX
ALU
Zero?
Branch taken
Next SEQ PC
EX / MEM
Memory
MUX
Next PC
MEM / WB
MUX
WB Data
IF
Instruction Fetch
ID
Instruction Decode Register Fetch
EX
Execute Address Calc.
MEM
Memory Access
WB
Write Back


next pc
+4
I-cache
fetch
pc + 4
pc + 8
I decode
r15
register read
instruction decode
immediate fields
mul
LDM/ STM
post-index
+4
shift
reg shift
pre-index
ALU
mux
execute
forwarding paths
B, BL
MOV pc
SUBS pc
byte repl.
load/store address
D-cache
buffer/ data
rot/sgn ex
LDR pc
register write
write-back