Digital Logic

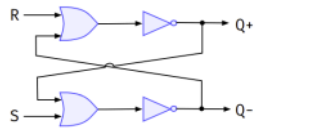
Clock cycles begin on rising edge and gates provide delay

Combinational Logic: Adders, Multiplexor

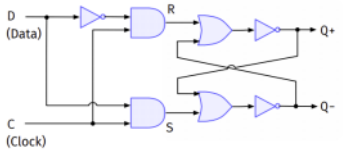
- Does not depend on itself, no loops
- Updates are nearly instant

Sequential Logic: SR Latch, D-Latch, flip-flop, write to reg

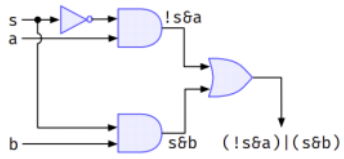
- State is dependent on itself, has loops
- Has delay and updates on rising edge



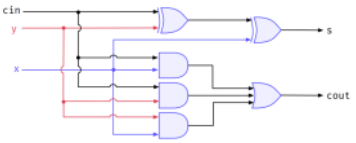
S	R	Q+	Q-	State
0	0	q	!q	Latched (Stored)
0	1	0	1	Resetting
1	0	1	0	Setting
1	1	–	–	Metastable/Error



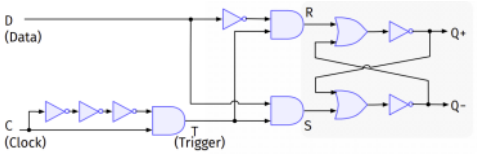
C	D	Q+	Q-	State
0	d	q	!q	Storing
1	d	d	!d	Latching



Multiplexor: to have more inputs must have $\lg N$ number of switch signal bits where N is the number of inputs



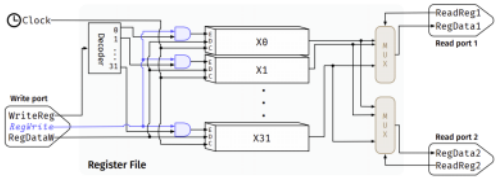
Adder: Adds bits x and y together and if there is a carry then the cout is 1. Same for the cin



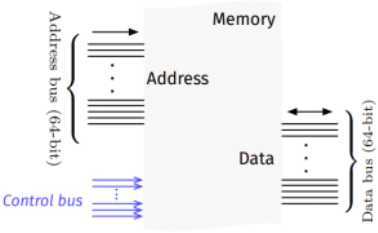
Allows for output to change only when C is on the rising edge

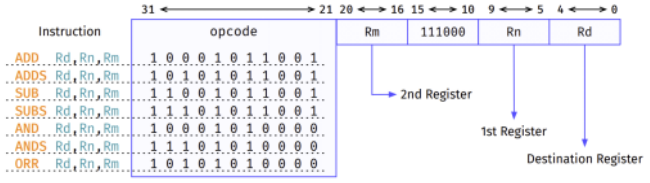
Microprocessor Design

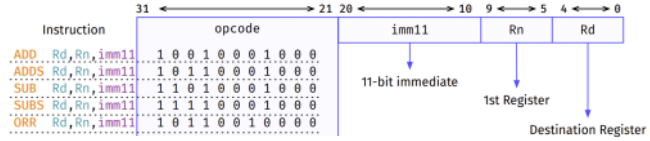
Reg File

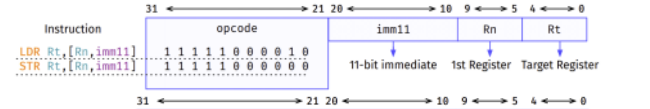


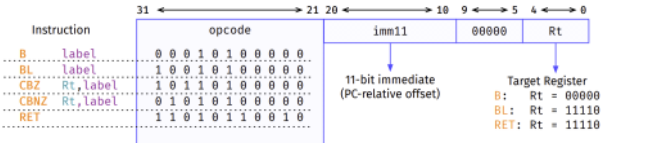
Memory

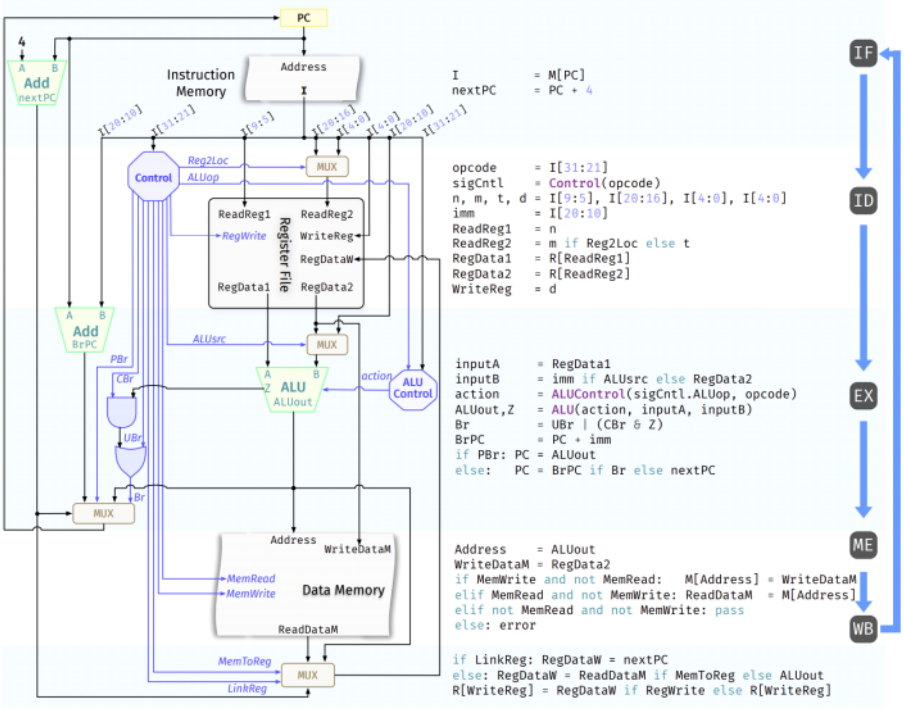




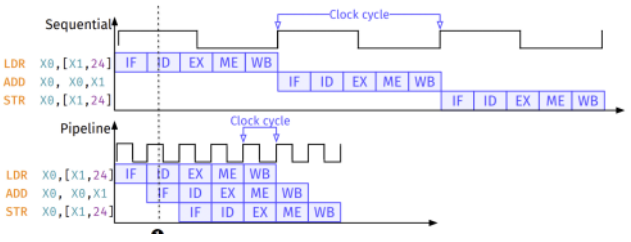








	ALUop	action	ALU operation
AND	10	0000	ALUout = inputA & inputB
ORR	10	0001	ALUout = inputA | inputB
SUB	10	0011	ALUout = inputA - inputB
ADD	10	0010	ALUout = inputA + inputB
LDR	00		
STR	00		
B	01	0111	pass inputB, *i.e.*, ALUout = inputB
BL	01		
CBZ	01		
RET	01		
ANDS	11	1000	ALUout = inputA & inputB (set Z)
ADDS	11	1010	ALUout = inputA + inputB (set Z)
SUBS	11	1011	ALUout = inputA - inputB (set Z)



1. **(IF) Instruction Fetching:** the instruction is obtained from memory;
2. **(ID) Instruction Decoding:** the fetched instruction will pass different fields to different data signals, and the `opcode` will be used for converting into control signals. Register data will also be read;
3. **(EX) Execution:** ALU will perform the operation based on the decoded instruction, and produce the result;
4. **(ME) Memory Access:** Sending data to memory or reading data from memory;
5. **(WB) Writing Back:** Write result back to register.

In none Pipe:

- CC: sum of all stages
- Latency: sum of all stages

In Pipelined:

- CC: length of longest stage
- Latency: number of stages multiplied by the CC

Performance Evaluation

Clock period: Length of clock cycle in ps (10^{-10})

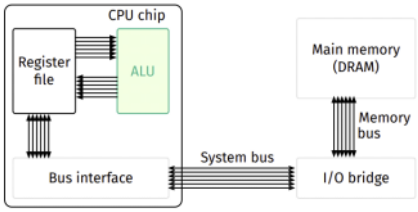
Clock Rate: Clock cycles every second in GHz $\frac{1}{clock\ period}$

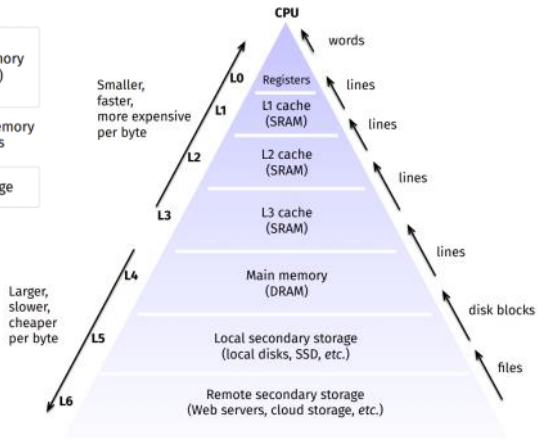
$$CPU\ time = I * CPI * C = \frac{(I)(CPI)}{CR}$$

I is the amount of instructions and *CPI* is the clock cycles per instruction. *C* is clock period

Cache type	What's cached?	Where is it cached?	Latency (cycles)	Managed by
Registers	4–8 bytes words	CPU core	0	Compiler
TLB	Address translations	On-Chip TLB	0	Hardware MMU
L1 cache	64-byte blocks	On-Chip L1	4	Hardware
L2 cache	64-byte blocks	On-Chip L2	10	Hardware
Virtual memory	4KB pages	Main memory	10^2	Hardware + OS
Buffer cache	Parts of files	Main memory	10^2	OS
Disk cache	Disk sectors	Disk controller	10^5	Disk firmware
Network buffer cache	Parts of files	Local disk	10^7	Web browser
Browser cache	Web pages	Local disk	10^7	Web browser
Web cache	Web pages	Remote server disks	10^9	Web proxy server

More formally, **spatial locality** means the data with nearby addresses tend to be used (referenced) close together in time, such as the array elements; **temporal locality** means recently referenced data are likely to be referenced again in the near future, such as `sum` in our example.





C: Capacity of Cache

C = S * E * B bytes

S = number of Sets

s = lg (S) : number of bits for set

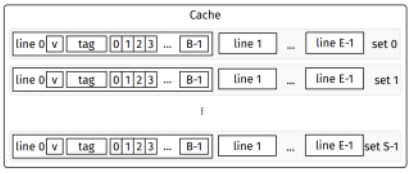
B = number of blocks

b = lg(B) : number of bits for offset

E = number of lines per set

t = remaining bits for tag

m = total bits in address

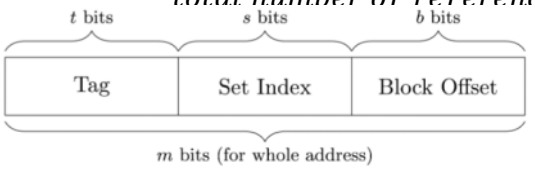


Direct mapped: E = 1

N-Way Associative: E = N

Fully Associative: S = 1

$$Miss\ rate = \frac{(number\ of\ caches\ missed)}{total\ number\ of\ references}$$



We have two ways to deal with write hit:

- **Write-through**: writes directly to the main memory as well as the cache;
- **Write-back**: updates the cache only first, and only updates the main memory when the line is replaced.

We also have two ways to deal with write miss:

- **Write-allocate**: copy the line into the cache from the main memory first, and then update the data in the cache;
- **No-write-allocate**: writes straight to the main memory without loading into the cache first.

Virtual Memory

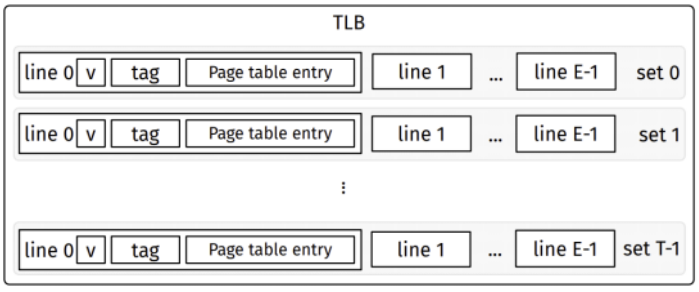
Virtual Address: Address received by cache from MMU which is a real memory address

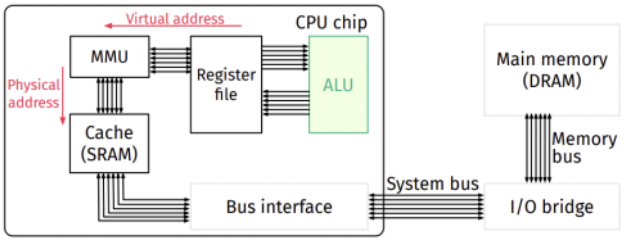
Translation:

Bits n-1..p+t	Bits p+t-1..p	Bits p-1..0
TLB tag	TLB index	Virtual page offset (VPO)

VPO = PPO

Virtual page number (VPN)





Virtual pages are smaller pieces of programs virtual memory space

Virtual memory is used since not all memory can be loaded at once.

Page hit if virtual memory address is mapped

Page fault if not possible

MMU gets virtual address and makes it physical address

TLB is special cache for pages that is stored in MMU

In an 8-way associative cache with 24 double-word blocks, where 16 bits are used for memory addresses. The cache is byte-addressed.

- To start off, we need to figure out the relevant information and fill in the cache table:
- E = 8 ; "8-way associative cache"
- m = 16 ; "16 bits are used for memory addresses"
- B = 8 ; "24 **double-word blocks**"
- S = 24/8 = 3 ; "24 double-word blocks", "8-way associative cache"

m	C	B	E	S	t	s	b
16		8	8	3			

In an 8-way associative cache with 24 double-word blocks, where 16 bits are used for memory addresses. The cache is byte-addressed.

- We can fill in the rest:
  - s = $[\log_2(S)]$ = 2
  - b = $\log_2(B)$ = 3
  - t = m – (s + b) = 11
  - C = B * E * S = 192

m	C	B	E	S	t	s	b
16	192	8	8	3	11	2	3

Data	# bits
nibble	4
byte	8
half word	16
word	32
double word	64
quadword	64