Dylan Faulhaber | CS 382-A | End-Term Exam Reference Sheet | 12/13/23

Digital Logic

Clock cycles begin on rising edge and gates provide delay

Combinational Logic: Adders, Multiplexor

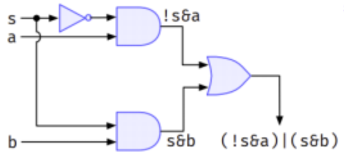
Does not depend on itself, no loops

Updates are nearly instant

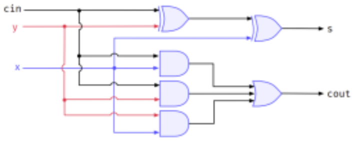
Sequential Logic: SR Latch, D-Latch, flip-flop, write to reg

State is dependent on itself, has loops

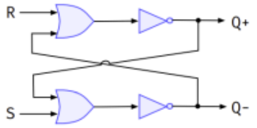
Has delay and updates on rising edge



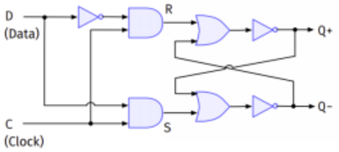
Multiplexor: to have more inputs must have lg *N* number of switch signal bits where N is the number of inputs



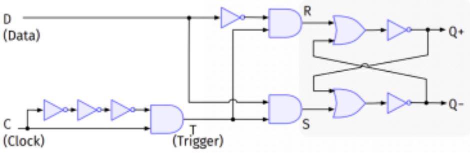
Adder: Adds bits x and y together and if there is a carry then the cout is 1. Same for the cin



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



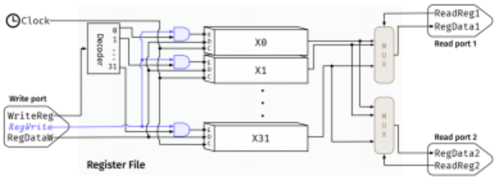
C	D	Q+	Q-	State
0	d	q	!q	Storing
1	d	d	!d	Latching



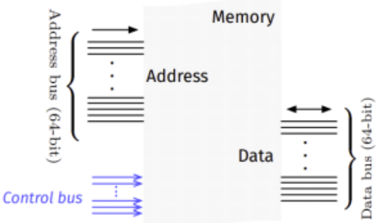
Allows for output to change only when C is on the rising edge

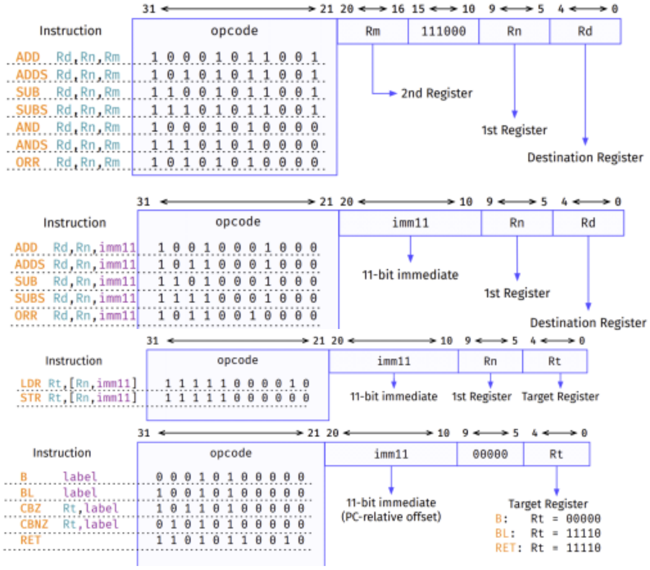
Microprocessor Design

Reg File



Memory



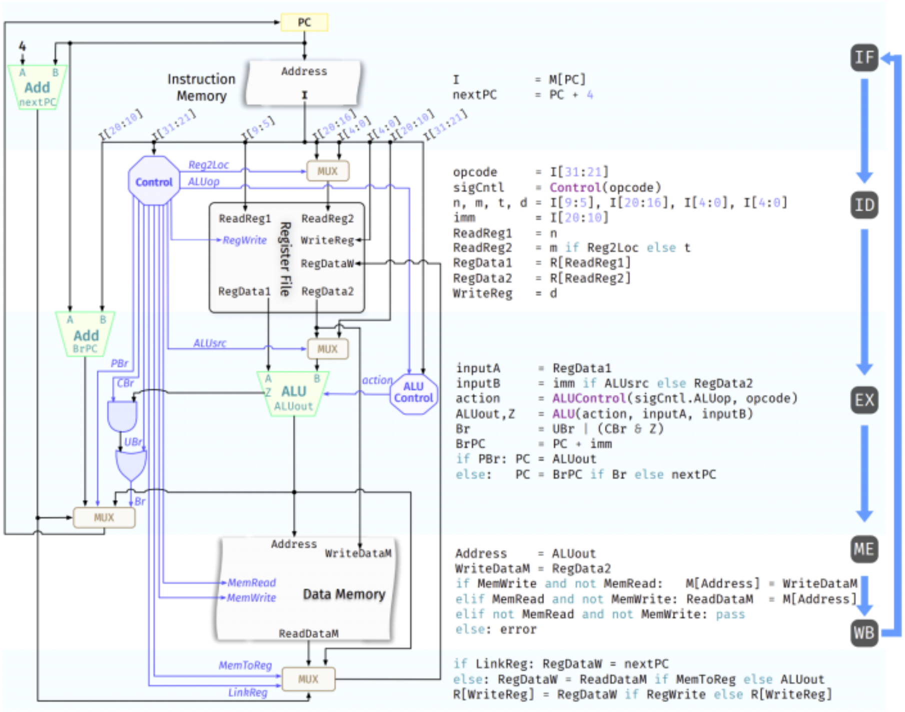


Instruction	opcode (31 ↔ 21)	Rm (20 ↔ 16)	111000 (15 ↔ 10)	Rn (9 ↔ 5)	Rd (4 ↔ 0)
ADD Rd,Rn,Rm	1 0 0 0 1 0 1 1 0 0 1				
ADDS Rd,Rn,Rm	1 0 1 0 1 0 1 1 0 0 1				
SUB Rd,Rn,Rm	1 1 0 0 1 0 1 1 0 0 1				
SUBS Rd,Rn,Rm	1 1 1 0 1 0 1 1 0 0 1				
AND Rd,Rn,Rm	1 0 0 0 1 0 1 0 0 0 0				
ANDS Rd,Rn,Rm	1 1 1 0 1 0 1 0 0 0 0				
ORR Rd,Rn,Rm	1 0 1 0 1 0 1 0 0 0 0				
		2nd Register		1st Register	Destination Register

Instruction	opcode (31 ↔ 21)	imm11 (20 ↔ 10)	Rn (9 ↔ 5)	Rd (4 ↔ 0)
ADD Rd,Rn,imm11	1 0 0 1 0 0 0 1 0 0 0			
ADDS Rd,Rn,imm11	1 0 1 1 0 0 0 1 0 0 0			
SUB Rd,Rn,imm11	1 1 0 1 0 0 0 1 0 0 0			
SUBS Rd,Rn,imm11	1 1 1 1 0 0 0 1 0 0 0			
ORR Rd,Rn,imm11	1 0 1 1 0 0 1 0 0 0 0			
		11-bit immediate	1st Register	Destination Register

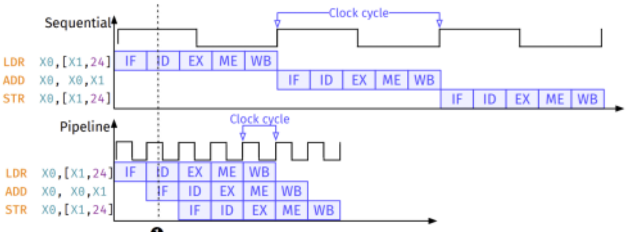
Instruction	opcode (31 ↔ 21)	imm11 (20 ↔ 10)	Rn (9 ↔ 5)	Rt (4 ↔ 0)
LDR Rt,[Rn,imm11]	1 1 1 1 1 0 0 0 0 1 0			
STR Rt,[Rn,imm11]	1 1 1 1 1 0 0 0 0 0 0			
		11-bit immediate	1st Register	Target Register

Instruction	opcode (31 ↔ 21)	imm11 (20 ↔ 10)	00000 (9 ↔ 5)	Rt (4 ↔ 0)
B label	0 0 0 1 0 1 0 0 0 0 0			
BL label	1 0 0 1 0 1 0 0 0 0 0			
CBZ Rt,label	1 0 1 1 0 1 0 0 0 0 0			
CBNZ Rt,label	0 1 0 1 0 1 0 0 0 0 0			
RET	1 1 0 1 0 1 1 0 0 1 0			
		11-bit immediate (PC-relative offset)		Target Register B: Rt = 00000 BL: Rt = 11110 RET: Rt = 11110



```
I           = M[PC]
nextPC      = PC + 4

opcode      = I[31:21]
sigCntl     = Control(opcode)
n, m, t, d = I[9:5], I[20:16], I[4:0], I[4:0]
imm         = I[20:10]
ReadReg1    = n
ReadReg2    = m if Reg2Loc else t
RegData1    = R[ReadReg1]
RegData2    = R[ReadReg2]
WriteReg    = d

inputA      = RegData1
inputB      = imm if ALUsrc else RegData2
action      = ALUControl(sigCntl.ALUop, opcode)
ALUout,Z    = ALU(action, inputA, inputB)
Br          = UBr | (CBr & Z)
BrPC        = PC + imm
if PBr: PC = ALUout
else:   PC = BrPC if Br else nextPC

Address    = ALUout
WriteDataM = RegData2
if MemWrite and not MemRead:   M[Address] = WriteDataM
elif MemRead and not MemWrite: ReadDataM  = M[Address]
elif not MemRead and not MemWrite: pass
else: error

if LinkReg: RegDataW = nextPC
else: RegDataW = ReadDataM if MemToReg else ALUout
R[WriteReg] = RegDataW if RegWrite else R[WriteReg]
```

	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