GATES ⊕ | CSC 258

(cf last page)

	BUFFER		AND		OR		XOR	
D⁻	Y = A, Y = B		Y = A·B		Y = A+B		Y = A ⊕ B	
A B: 00, 01, 10, 11	A (B): 0 0 1 1	$\overline{A}$ ($\overline{B}$): 1 1 0 0	AB: 0 0 0 1	$\overline{AB}$: 1 1 1 0	A + B: 0 1 1 1	$\overline{A+B}$: 1 0 0 0	A⊕B: 0 1 1 0	A <=> B: 1 0 0 1
D⁺	$Y = \overline{A}$, $Y = \overline{B}$		$Y = \overline{A \cdot B}$		$Y = \overline{A+B}$		$Y = \overline{A \oplus B}$	
⊖	NOT		NAND		NOR		XNOR	

CIRCUITS

ABC	Minterms			Maxterms		
000	$\overline{A}\overline{B}\overline{C}$	m_0	000 → 1	A + B + C	M_0	000 → 0
001	$\overline{A}\overline{B}C$	m_1	001 → 1	A + B + $\overline{C}$	M_1	001 → 0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
110	$AB\overline{C}$	m_6	110 → 1	$\overline{A}$ + $\overline{B}$ + C	M_6	110 → 0
111	ABC	m_7	111 → 1	$\overline{A}$ + $\overline{B}$ + $\overline{C}$	M_7	111 → 0

SOM
Every min in the combi is high

POM
Every Max in the combi is low

KARNAUGH

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	2	4	3
$\overline{A}B$	5	6	8	7
AB	13	14	16	15
$A\overline{B}$	9	10	12	11

Minterms — can fill from maxterms by 'converting' 0→1

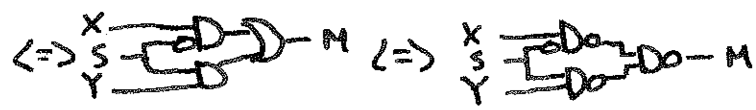
built such that
- differ by one
- order can be

i.e. mentally linked to 0000 - 0001 - 0010...

DEVICES

MUX (·S· Demux)

$M = \overline{S}X + SY$ <=>



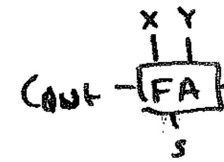
ADD

HA:

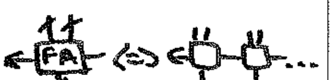
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

S → X⊕Y
C → X·Y

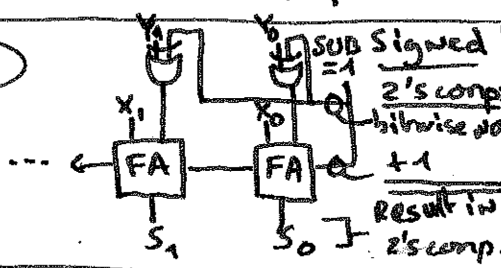
FA:



$C_{out} = XC_{in} + YC_{in} + XY$
$S = X \oplus Y \oplus C_{in}$



SUB



Signed: 2's comp: bitwise not, +1, result in 2's comp.

Unsigned: sign bit | NUMBER; 1 → ⊕, 0 → ⊖

X−Y: (without sign bits) X + 2's comp(Y)

Result: [ ] [R]; 1 → [1] R; 0 → [0] 2's comp(R)

Note on 2's comp.

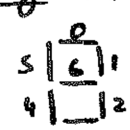
'divides' scope of possible nbs given by 2:

	unsigned	signed
000	0	0
001	1	1
...	...	...
100	4	−4
...	...	...
111	8	−1

Thus most negative number is 1000...

DEC

e.g. 7 seg disp (HEX0) → 0, 2, 3, 5, 6, 7, 8, 9



ABCD
0000 → 0
0001 → 1
⋮
1001 → 9
... → don't care

	$\overline{C}\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
$\overline{A}\overline{B}$	1	0	1	1
$\overline{A}B$	0	1	1	1
AB	X	X	X	X
$A\overline{B}$	1	1	X	X

$HEX0 = A + C + BD + \overline{B}\overline{D}$

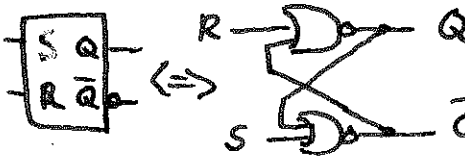
Combinational circuits : ∅ FEEDBACK => OUTPUT DEPENDS ON INP.

Sequential circuits : FEEDBACK => VALUES ARE STORED IN THE CIRCUIT

SEQCIR

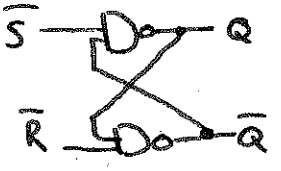
Simple (set/reset) latches

SR [NOR] latch



S	R	Q_{next}	Action
0	0	Q	hold
0	1	0	Reset
1	0	1	Set
1	1	X	N/a

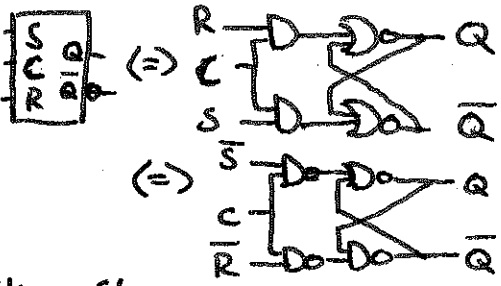
$\overline{SR}$ [NAND] latch



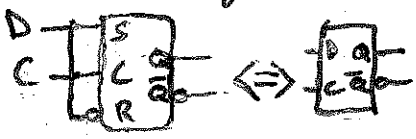
$\bar{S}$	$\bar{R}$	Q_{next}	Action
0	0	X	N/a
0	1	1	Reset
1	0	0	Set
1	1	Q	Hold

Gated (clocked) latches

Clocked SR latch



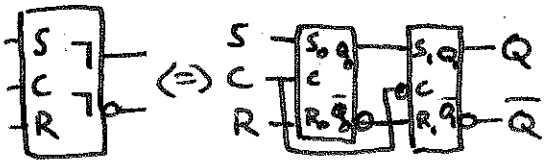
D latch (gated)



C	D	Q	$\bar{Q}$	Action
0	X	Q_p	$\bar{Q}_p$	hold
1	0	0	1	Reset
1	1	1	0	Set

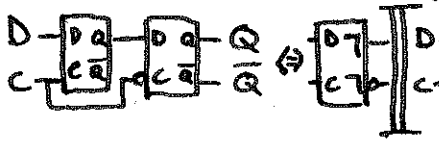
Flip-flops

SR (m/s) FF



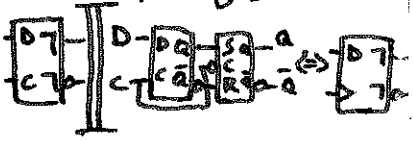
⎍ trig SR

D (m/s) FF



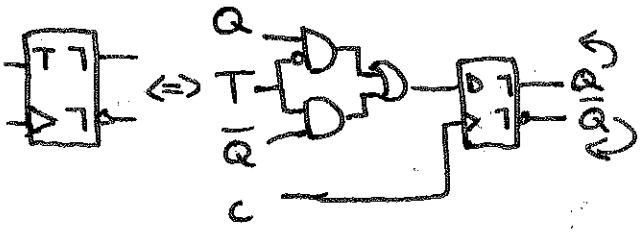
⎍ trig D

D (posedge) FF



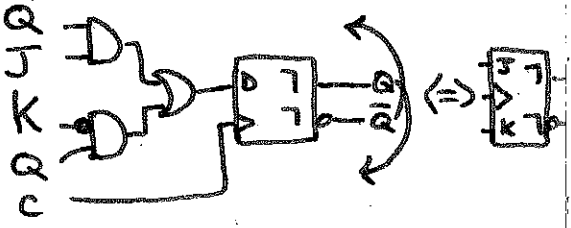
⌐ trig D

T FF



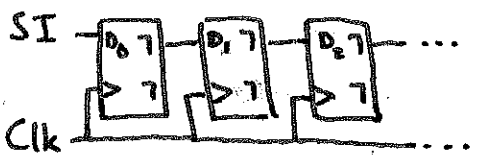
$Q_n = T \oplus Q_p$

JK FF

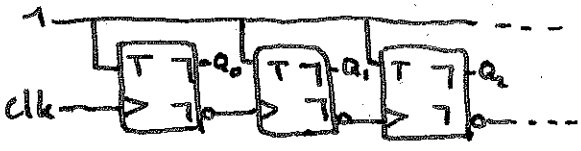


$Q_n = J\bar{Q}_p + \bar{K}Q_p$

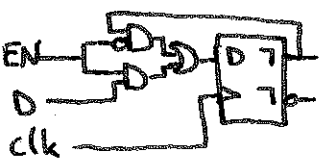
Shift register



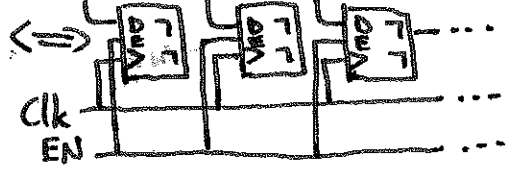
Asynchronous (ripple) Counter



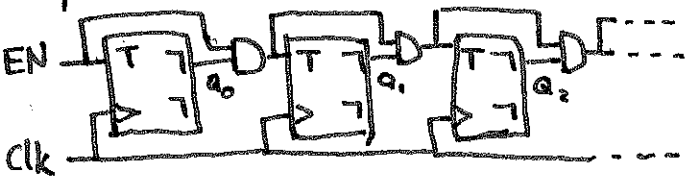
D FF with enable:



Load Register



Synchronous Counter

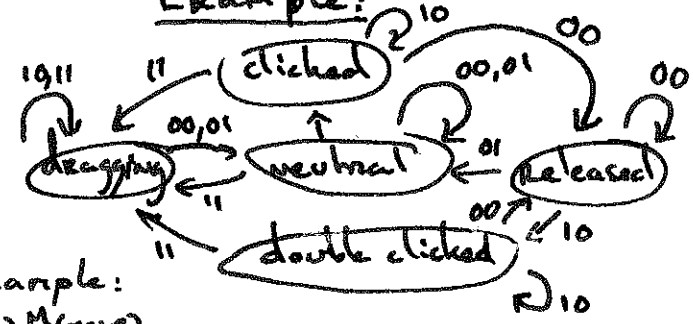


FSM - Finite State Machine : abstract model that captures the operation of a sequential circuit

- FSM Design:

1. Draw state diagram
   - → encompasses all the (states) and transitions [→] between states that relate to signal input (00, 01 ...) → ie example: P(press) M(move)
2. Derive state table
   - → straight forward from diagram
3. Assign Flip-flops
   - → nb ff = $\lceil \log_2$ nb states$\rceil$
   - → avoid bad transitions
     - Minimize nb
     - avoid changing more than 2 values at once
     - intermediary states must be 'harmless' (→ 'don't care' output → unused)
4. Redefine state table
   - → Replace state names by FF values
5. Design combinational circuit
   - → draw karnaugh maps for each flip-flop and derive FF bool equa°
   - → draw circuit from FF boolean equa°

Example:



State$_P$ (previous)	P	M	State$_n$ (current)
neutral	1	0	clicked
clicked	1	0	clicked
⋮	⋮	⋮	⋮

$\lceil \log_2 5 \rceil = 3$

FF$_0$	FF$_1$	FF$_2$	P	M	FF$_0$	FF$_1$	FF$_2$
0	0	0	1	0	1	0	0
1	0	0	1	0	1	0	0
...							

	$\bar{F}_2\bar{F}_1\bar{F}_0\bar{P}\bar{M}$	$\bar{F}_2\bar{F}_1\bar{F}_0\bar{P}M$	$\bar{F}_2\bar{F}_1\bar{F}_0P\bar{M}$	$\bar{F}_2\bar{F}_1\bar{F}_0PM$	$\bar{F}_2\bar{F}_1F_0\bar{P}\bar{M}$	$\bar{F}_2\bar{F}_1F_0\bar{P}M$	$\bar{F}_2\bar{F}_1F_0P\bar{M}$	$\bar{F}_2\bar{F}_1F_0PM$
$\bar{F}_2F_1$	0	0	0	0	1	x	0	1
$F_2\bar{F}_1$	1	x	0	0	x	x	x	x
F_2F_1	x	x	x	x	1	0	x	1
$\bar{F}_2F_1$	0	0	0	0	x	x	x	x

$$F_0 = F_2\bar{P} + F_0P\bar{M} + F_0\bar{P}\bar{M}$$

- **FSM types:**
  - → Moore Mach: output depends solely on state
  - → Mealy Mach: output depends on state and input

PROCESSORS

- **Booth's multiplication algorithm**
  - → $m \times R = p$ (m has x bits, R has y bits, P has x+y+1 bits)
  - → A = m, (y+1) 0s
  - → S = −m (2's comp), (y+1) 0s
  - P = x 0s, R, 0
  - → loop y times:
    - → compare last 2 bits of P
      - 00, 11 → P
      - 01 → P+A
      - 10 → P+S
    - → arithmetically shift the value 1 bit to the right
  - → drop the least significant bit from P, the result

- → example: $\begin{cases} 3 \times (-4) = -12 \\ x = y = 4 \end{cases}$

→ A =	0011	0000	0
S =	1101	0000	0
P =	0000	1100	0

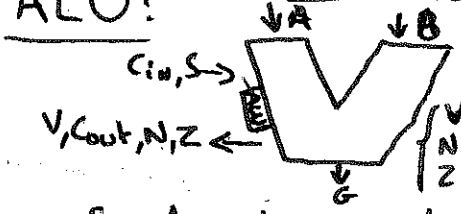
→ 1. P = 000011000 ⇒ P = 000001100
2. P = 000001100 ⇒ P = 000000110
3. P = 000000010 ⇒ P+S = 111010110
   P = 111010011
4. P = 111010011 ⇒ P = 111101001

→ P = 11110100 (2's comp of −12)

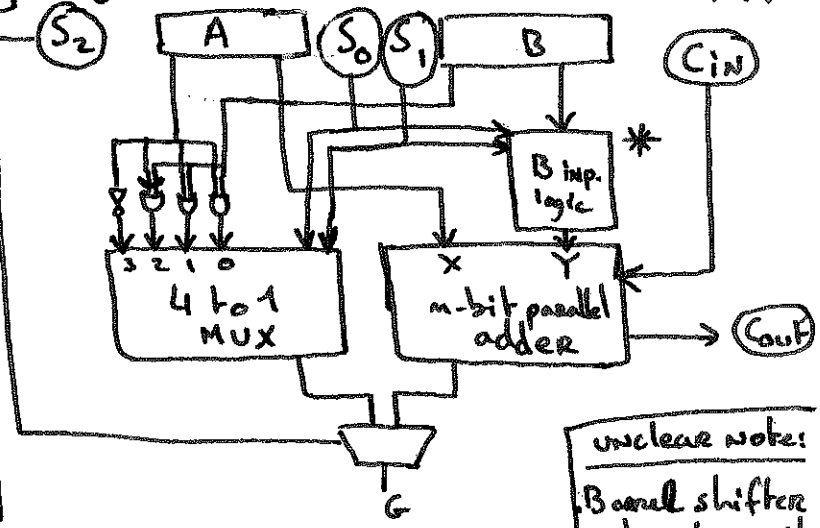
Note: this won't work for 'most negative number' in 2's comp ⇒ add a 0 bit to left

• ALU:

overview | more detail



Logical unit — Arithmetic Unit

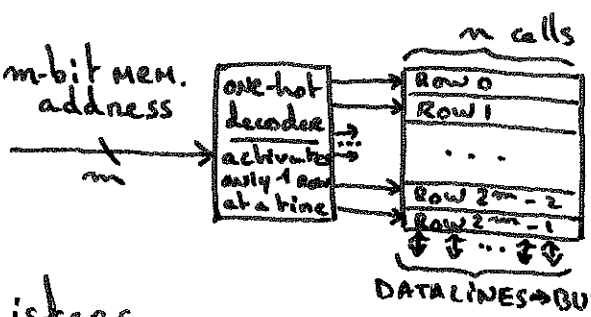


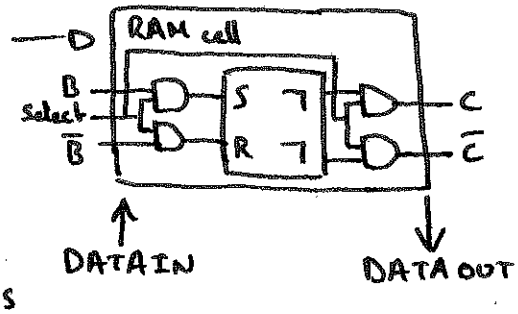
- S_2 : A-mode / L-mode select
- S_0, S_1 : Operation select

*	S_1	S_0	Y	$C_{IN}=0$	$C_{IN}=1$
	0	0	All 0s	A (transfer)	A+1 (incr.)
	0	1	B	A+B (add)	A+B+1
	1	0	$\bar{B}$	A+$\bar{B}$	A+$\bar{B}$+1 (sub)
	1	1	All 1s	A-1 (decr.)	A (transfer)

unclear note:
Barrel shifter acts alongside ALU

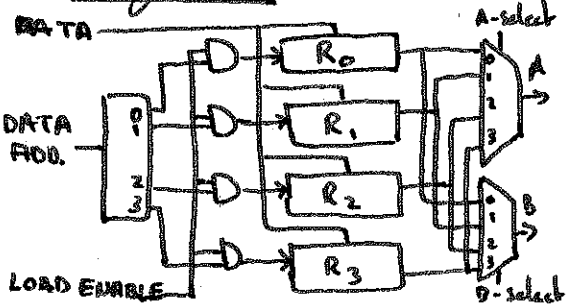
• Memory





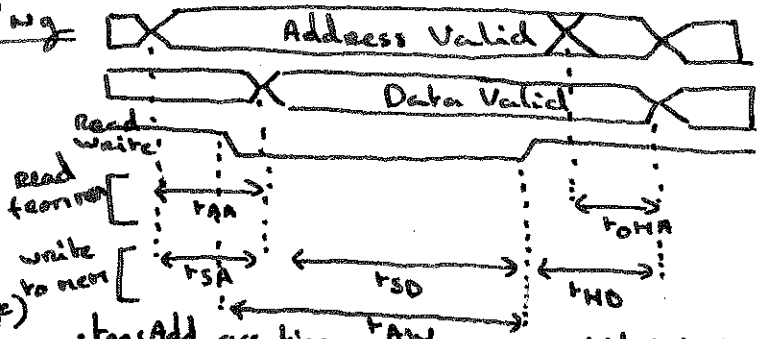
- Some buses are used as i/o, thus ▷ WE tristate buffer
- Word select: Row #
- Bit select: cell #

• Registers



- Registers (few compared to mem) host data during instructions to be used by those instructions.

• Timing



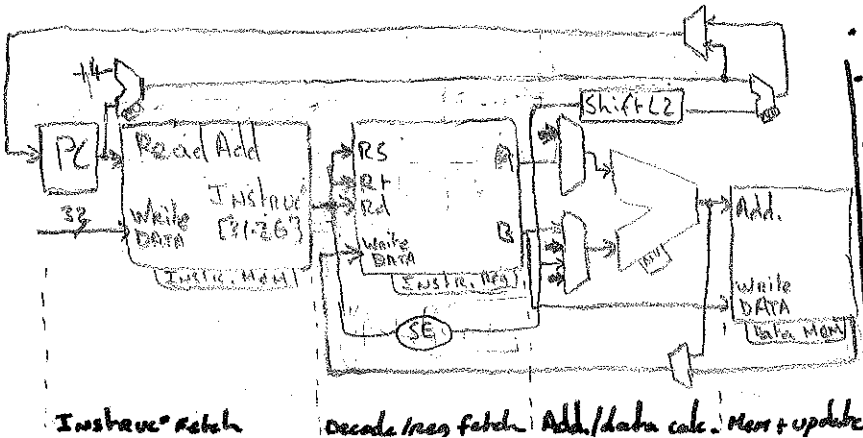
- t_{AA}: Add. acc. time → address values stable before reading
- t_{OHA}: Out. hold add. → hold data high after add. value change
- t_{SA}: Setup add. time → address values stable before write
- t_{AW}: Add. width time → write signal high
- t_{SD}: Setup data time → data values setup at deactivation
- t_{HD}: Hold data time → data remains high after write

• Buses

- wires allowing communication between components (multiple can read, only 1 can write)
- each component connects to the bus with a tri-state buffer (high impedance signal when not reading or writing)

Bus driver: component that reads from bus

• MIPS Datapath



Instruc° fetch | Decode/reg fetch | Add./data calc. | Mem + update

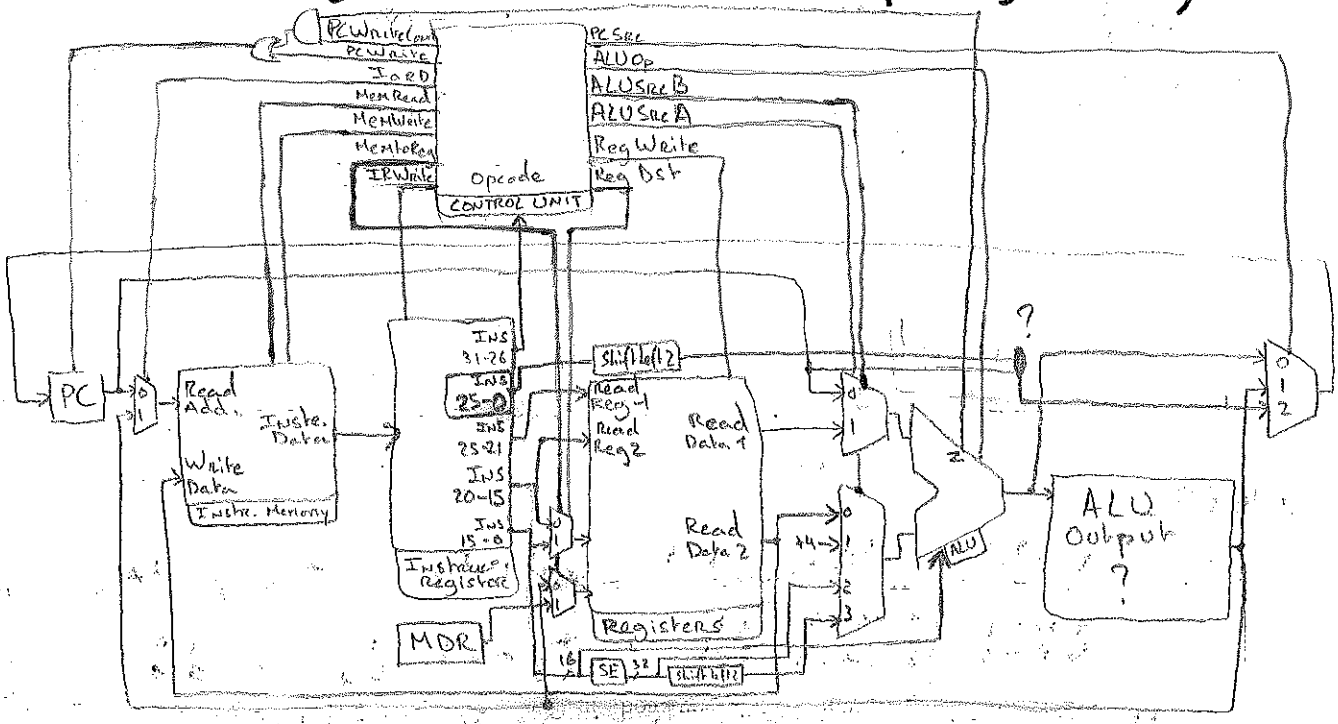
• MIPS Instruc°

- R-type | opcode (31–26) | RS (25–21) | RT (20–16) | Rd (15–11) | shamt (10–6) | funct (5–0)
  Reg-Reg → read RS and RT, feed to ALU, update Rd
  opcode = 000000
- I-type | opcode (31–26) | RS (25–21) | RT (20–16) | immediate (displacement) (15–0)
  load/store → Read RS (RT), feed to ALU (imm); memory req
  branch → Read RS and RT, ALU compare, update/backup PC
- J-type | opcode (31–26) | address (25–0)
  jump → loads add. in PC (after optional backup)

Control Unit

→ FSM whose input is opcode, outputs are the various muxes/etc. that channel the flow of info

- PCWrite: ALU → PC
- PCWriteCond: ALU → PC iff Zero
- I or D: Instr. (PC) or Data (ALU)
- MemRead: CPU reading Mem
- MemWrite: CPU writing Mem
- MemtoReg: Mem → Reg (or ALU → Reg)
- IRWrite: Instr. Reg ← Instr. Mem
- PCSource: (Jump or ALU op) → PC
- ALUOp (3w): Alu op.
- ALUSrcA: (PC or Reg) A → ALU
- ALUSrcB: B → ALU
- RegWrite: CPU writing Reg
- Reg Dst: which part of Instr. (rt or rd) is providing add. for RegWrite



ASSEMBLY

- MIPS → Microprocessor without Interlocked Pipeline Stages
  - → reg-to-reg: All ops. are done on data in regs.
  - → 32 regs + 3 regs (PC, HI, LO) → for mult.
    - 0 → $zero
    - 1 → $at (assembler)
    - 2-3 → $v0, $v1 (return val.)
    - 4-7 → $a0-$a3 (func' args)
    - 8-15, 24-25 → $t0-$t9 (temp)
    - 16-23 → $s0-$s7 (saved temp)
    - 26,27 → $k0, $k1 (interrupt/trap)
    - 28 → $gp (global ptr), $sp (stack ptr)
    - 29 → $fp (frame ptr), $ra (ret. add.)

Notes!
i: Immediate
u: unsigned
SE: sign extend
16 bits → 32 bits (first bit is rep. if signed)
ZE: zero extend (first 16 bits are 0s)

- Instruc'
  - → Arithmetic (add, addi (I-TYPE), div, mult, sub) e.g. addi: $t = $s + SE(i)
  - → Logic (and, andi, nor, or, ori, xor, xori) e.g. andi: $t = $s and ZE(i)
  - → Shift (sll, sllv, sra, srav, srl, srlv) e.g. sllv: $d = $t << $s
    [s: shift, l: left, r: right, l: logic, a: arithm, v: variable wrt ($s)]
  - → Comparison (slt, slti) [stores 1 if $s is less] e.g. slti: $d = ($s < SE(i))
  - → Branch (beq, bgtz, blez, bne) e.g. bne: ($s != $t) ⇒ pc += i << 2
    [takes label as input, used to compute i = [label - (curr PC + 4)] >> 2]
  - → Jump (j, jal, jr, jalr) [jr, jalr: not J-TYPE] eg. jr: pc = $s
    [jal: jump and link → curr PC → $ra] jal: $ra = PC
    PC += i << 2
  - → Load/Store (lb, lh, lw, sb, sh, sw) e.g. lhu: $t = ZE(MEM[$s + i]:2)
    [b: byte (8b), h: half word (4 bytes), w: word (2w)]
  - → Data mov (mfhi, mflo, mthi, mtlo)
  - → Trap [trap instr. e.g. print_int, exit]
- Syntax
  - → program starts with main:, .data: data declara', .text: start prog. instructions
  - → var. declara': label .type value [type: .word, .byte, .space ...]

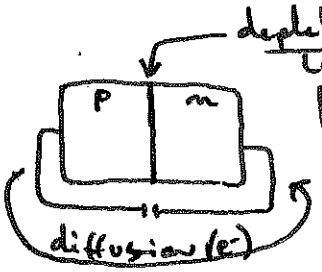
TRANSISTORS

p-n junctions

col.	13	14	15
e^-	3	4	5
elem	B	C	N
	Al	Si	P
		Ge	

doping: ▲ p-type SC+13; ▼ n-type SC+15

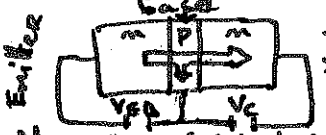
semiconductors



depletion layer → creates field:
- holes → p } drift
- e^- → n
- prevents e^- from getting through

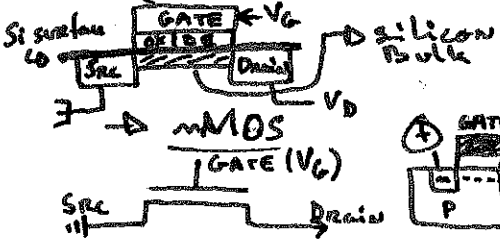
- Forward bias: ⊕ p | n ⊖ → depl layer ↓, e^- diffusion ↗
- Reverse bias: ⊖ p | n ⊕ → short circuit

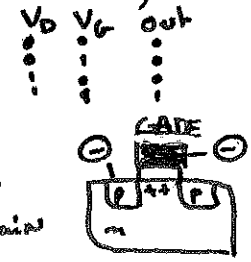
• BJT (Bipolar Junction Transistors)



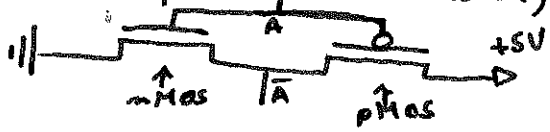
V_{EB}	V_C	Out
Hi	Hi	Lo
Hi	Lo	Lo
Lo	Hi	Lo
Lo	Lo	?

• MOSFET (Metal Oxide Semiconductor Field Effect Transistors)





• CMOS (Compl. Metal Oxide Semicond.)



A=0 ⇒ nMOS channel ⊕ (resistance) / pMOS channel ⊖ (∞ res) ⇒ +5V → $\bar{A}$ ‖ GRND ⇒ $\bar{A}$ = 1

A=1 ⇒ +5V ‖ $\bar{A}$ → GRND ⇒ $\bar{A}$ = 0

GATES

(cf 1st page)

AB	BUF	NOT	AND	NAND	OR	NOR	XOR	NXOR
	A→Y, B→Y	A→Y, B→Y	A, B → Y	A, B → Y	A, B → Y	A, B → Y	A, B → Y	A, B → Y
	Y=A, Y=B	$Y=\bar{A}$, $Y=\bar{B}$	Y=AB	$Y=\overline{AB}$	Y=A+B	$Y=\overline{A+B}$	$Y=A\oplus B$	$Y=\overline{A\oplus B}$
0 0	0	1	0	1	0	1	0	1
0 1	0	1	0	1	1	0	1	0
1 0	1	0	0	1	1	0	1	0
1 1	1	0	1	0	1	0	0	1