Computer Hardware Cheatsheet

Multi-input XOR: $n(1) \in 2^{N+1} \rightarrow$ output is 1.

Flip Flops:

SRFF

Q^t	S	R	Q^{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	K

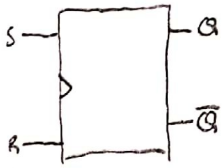
$S=R=0 \rightarrow$ Keep Q^t

$Q(t+1) = Q(t)\cdot\overline{R(t)} + S(t)$

$Q^{(t)}$ $Q^{(t+1)}$	00	01	10	11
S R	0Ø	10	Ø0	01

$Q^{(t)}$ \ SR	00	01	11	10
0	0	0	K	1
1	1	0	K	1

$Q^{(t+1)} = S + \overline{R}\cdot Q^{(t)}$



JKFF

Q^t	J	K	Q^{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

$Q^{t+1} = Q^t\cdot\overline{K(t)} + Q^t\cdot J(t)$

$J=K=0 \rightarrow$ Keep Q^t

$J=K=1 \rightarrow \neg Q^t$

Q^t Q^{t+1}	00	01	11	10
S R	0Ø	1Ø	Ø0	Ø1

Q^t \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$Q^{t+1} = J\cdot Q^t + \overline{K}\cdot Q^t$

DFF

Q^t	D	Q^{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

$Q^{t+1} = D(t)$

Q^t Q^{t+1}	00	01	11	10
D	0	1	1	0

Q^t \ D	0	1
0	0	1
1	0	1

$Q^{t+1} = D$

TFF

Q^t	T	Q^{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

$T=0 \rightarrow$ Keep Q^t

$T=1 \rightarrow \overline{Q^t}$

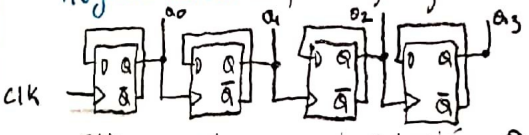
Q^t Q^{t+1}	00	01	11	10
T	0	1	0	1

Q^t \ T	0	1
0	0	1
1	1	0

$Q^{t+1} = Q^t \oplus T$

Counters:

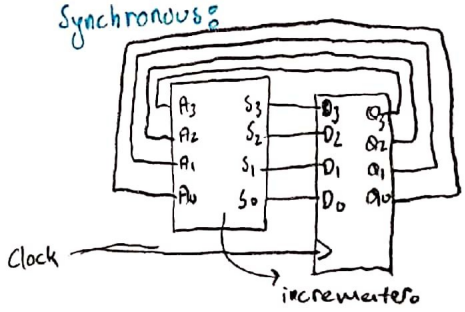
Asynchronous: Output is free from the clock signal.



This example can count 15 to 0.

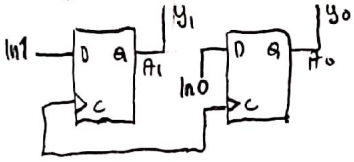
If input were $\overline{Q}$, example counts 0 to 15.

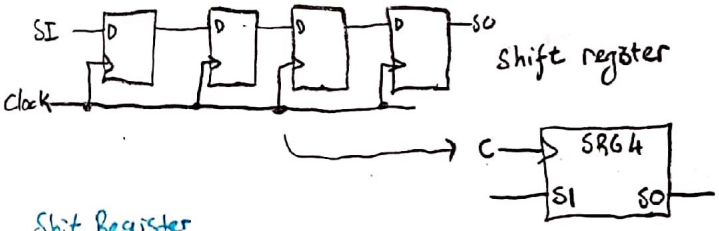
Synchronous:



Registers:

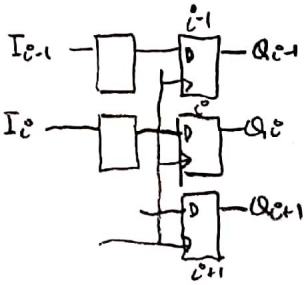
Register storages a vector of binary values.





Shift Register with Parallel Load:

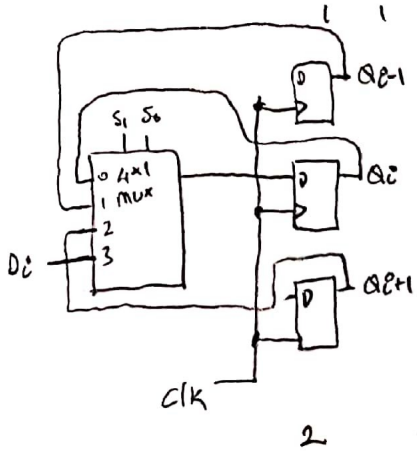
Shift	Load	operation
0	0	No change
0	1	Load Parallel Data
1	X	Shift Left



$D_i = \overline{shift}\ \overline{load}\ Q_i + \overline{shift}\ load\ I_i + shift\ Q_{i-1}$

Bidirectional Shift Register:

S_1	S_0	operation
0	0	No change
0	1	Shift Left
1	0	Shift right
1	1	Parallel load



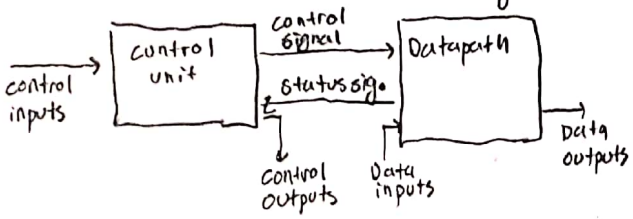
Register Transfer Operations

The movement and processing of data stored in registers.

There are three basic components
- → set of registers
- → operations
- → control of operations

And the system partitioned into 2 types of modules:
- → Datapath: performs data processing op.
- → Control unit: determines sequence of op.

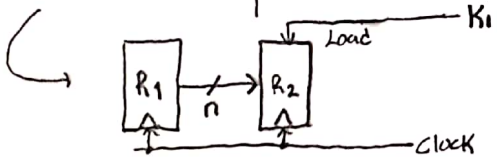
Datapaths are defined by their registers and the operations performed on binary data stored in the registers.



Conditional Transfer

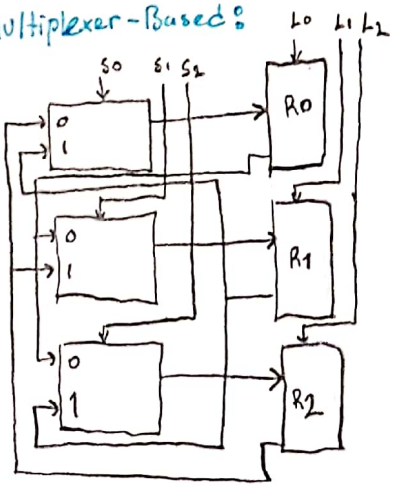
if (K1 == 1):
R2 ← R1

Shortened to
K1: R2 ← R1



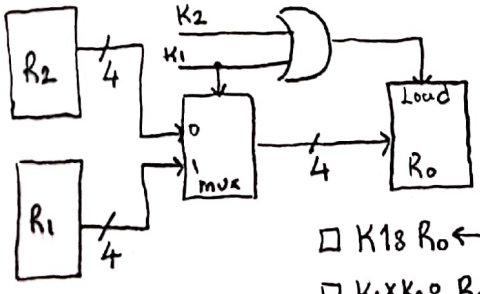
Register Transfer Structures

Dedicated-Multiplexer-Based:



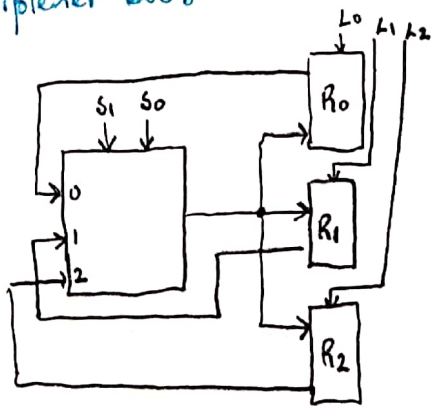
	Select			Load		
	S_0	S_1	S_2	L_0	L_1	L_2
$R_0 \leftarrow R_2$	0	X	X	1	0	0
$R_2 \leftarrow R_1, R_1 \leftarrow R_0$	X	0	1	0	1	1

MUX-Based:



- ☐ K1: $R_0 \leftarrow R_1$
- ☐ K1 K2: $R_0 \leftarrow R_2$

Multiplexer Bus:



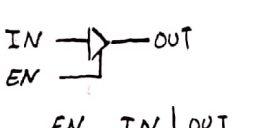
Single bus

	Select		Load		
	S_1	S_0	L_2	L_1	L_0
$R_0 \leftarrow R_2$	1	0	0	0	1
$R_0 \leftarrow R_1, R_2 \leftarrow R_1$	0	1	1	0	1

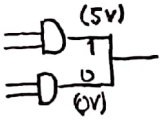
$R_0 \leftarrow R_1, R_1 \leftarrow R_0$ Impossible

Three State Bus:

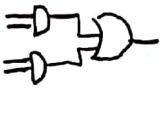
The 3-input MUX can be replaced by a 3-state bus and 3-state buffers.



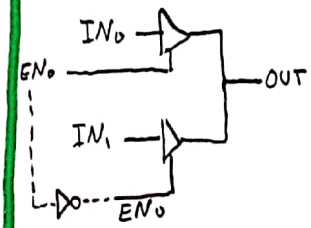
EN	IN	OUT
0	X	Hi-z
1	0	0
1	1	1



=> Not a short circuit



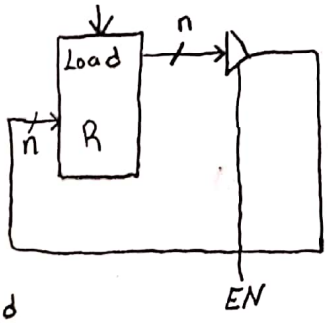
=> This or three state buffer

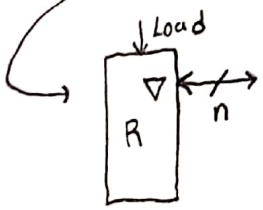


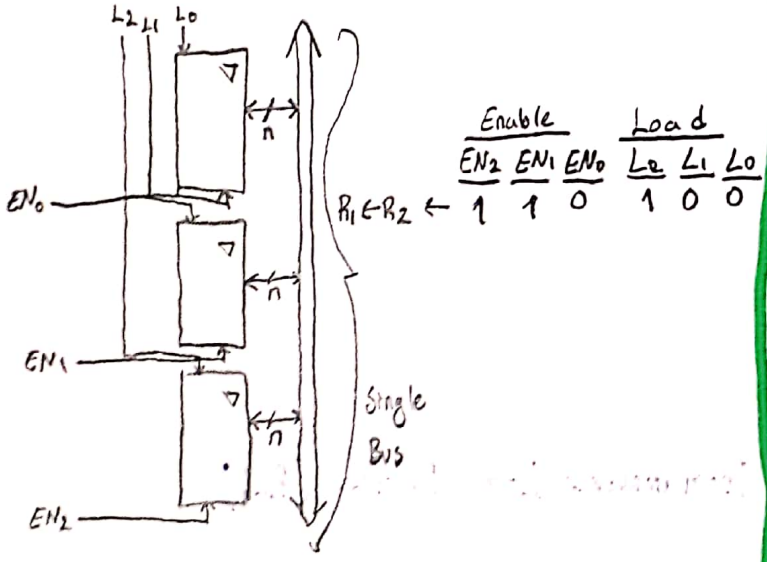
EN_1	EN_0	IN_1	IN_0	OUT
0	0	X	X	Hi-z
0	1	X	0	0
0	1	X	1	1
1	0	0	X	0
1	0	1	X	1
1	1	0	0	0
1	1	0	1	~
1	1	0	1	~
1	1	1	1	1

- → cost is further reduced, but transfers are limited.
- → Characterized the simultaneous transfers possible with this structure.
- → Characterized the cost savings and compare.

Registers with Bidirectional I/O Lines



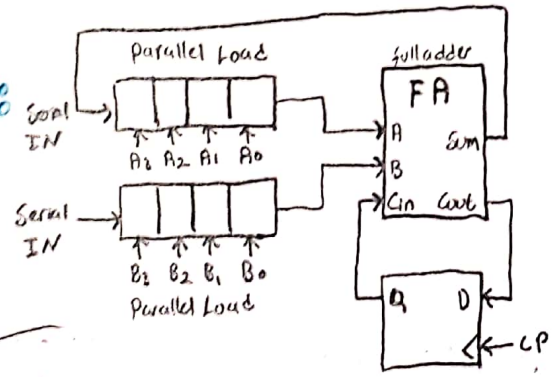


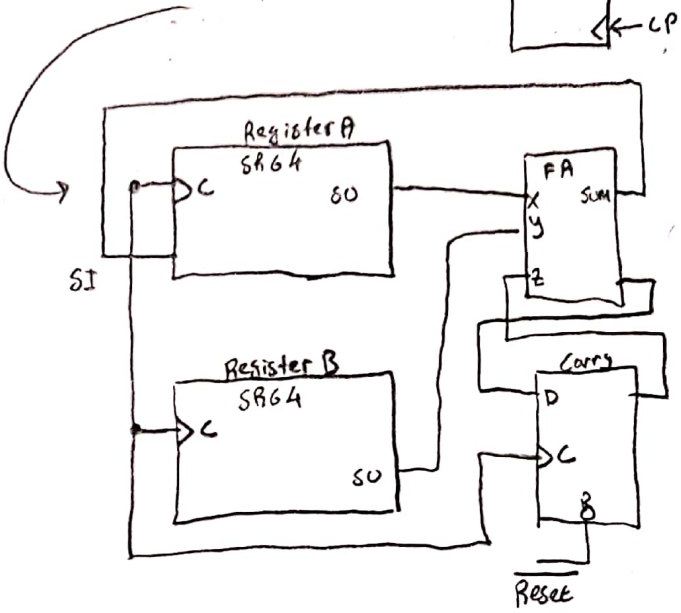


Enable			Load		
EN_2	EN_1	EN_0	L_2	L_1	L_0
1	1	0	1	0	0

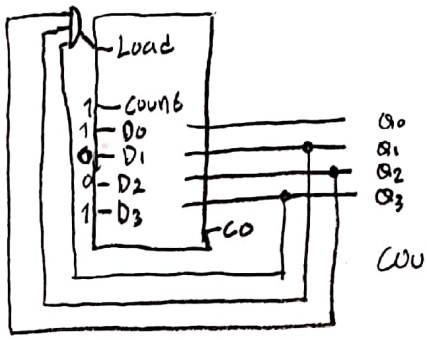
$R_1 \leftarrow R_2$

Serial Adder



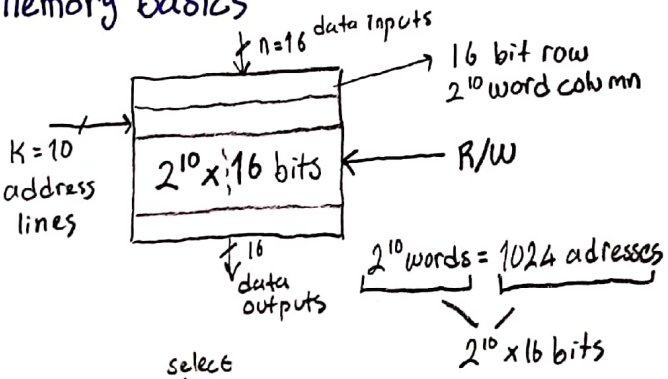


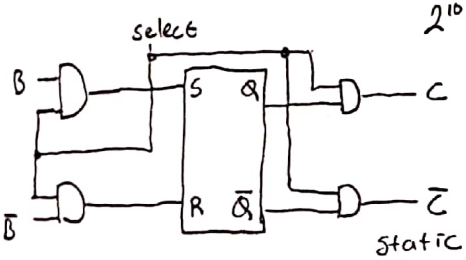
A BCD Counter Example



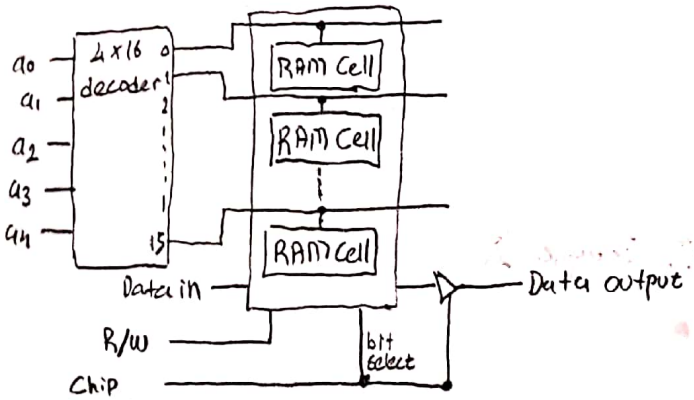
counts [9, 14]

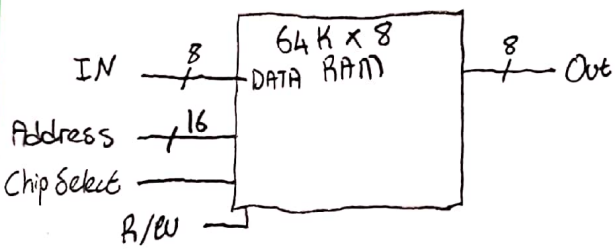
Memory Basics





static RAM cell



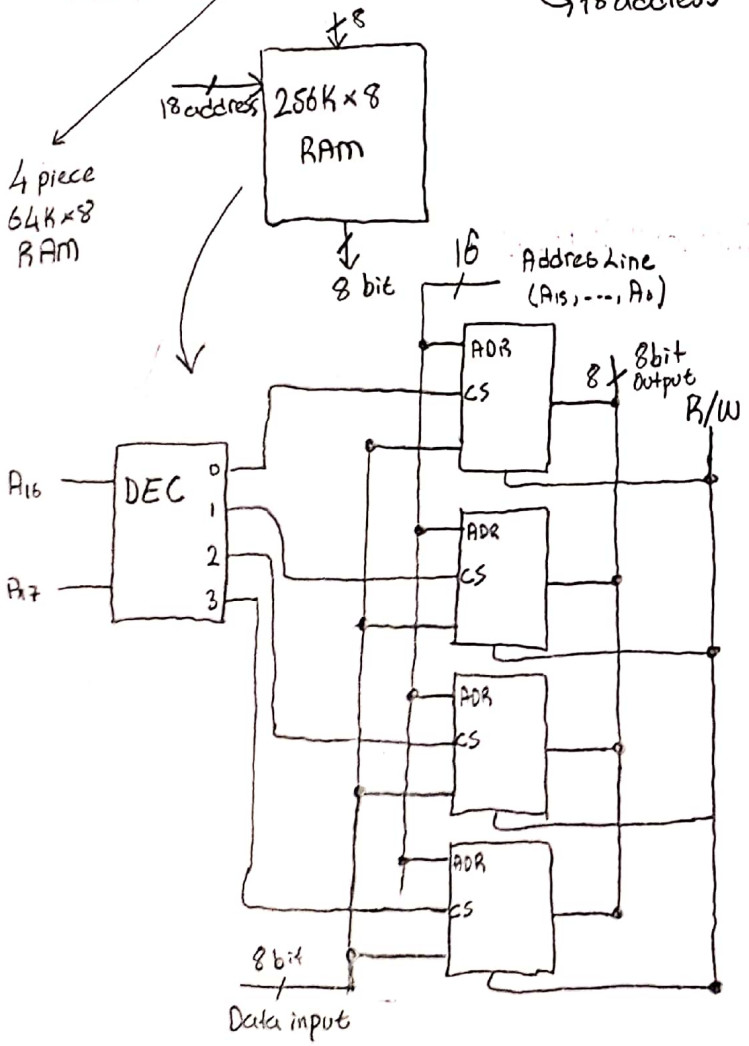


RAM Example 1

- Design 256K x 8 RAM by using 64K x 8 RAM

$\frac{256K \times 8}{64K \times 8} = 4$ → data

$2^8 = 256 \rightarrow 2^{(8+10)} = 256K$ → 18 address

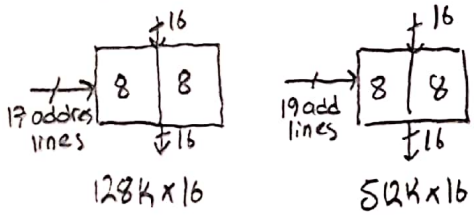


RAM Example 2

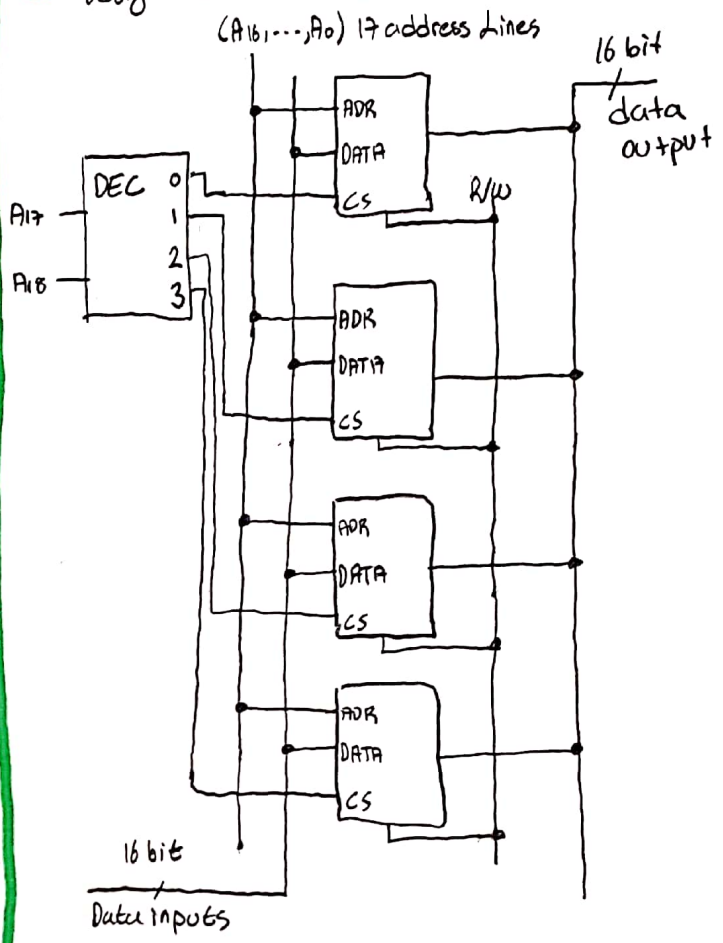
- How many 128K x 16 RAM chips are needed to provide a memory capacity of 1MB?

$1MB = n \cdot 16\,bit \rightarrow n = 512K$

$\frac{512K \times 16}{128K \times 16} = 4$ piece of 128K x 16 RAM



- Design with decoder.



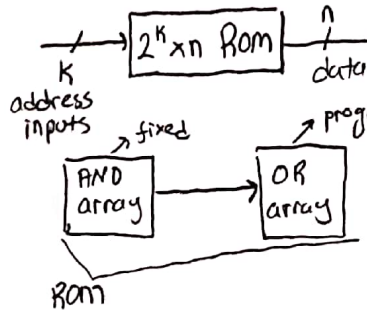
Programmable Logic Devices (PLD)

PLD's are fabricated with structures that

- implement logic functions
- used to control connections to store information specifying the actual logic functions

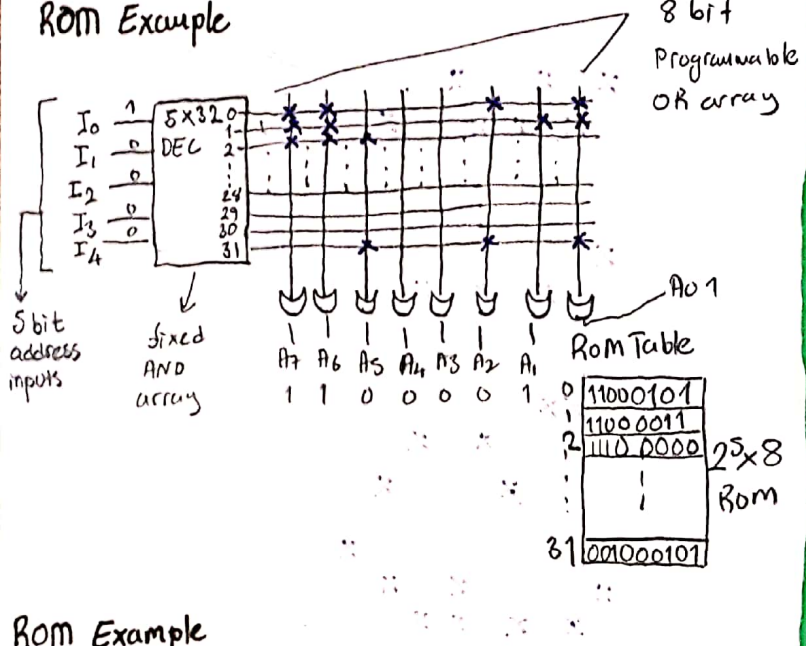
- ROM
- PAL
- PLA
- CPLD/FPGA

Prom (Programmable Read Only Memory):



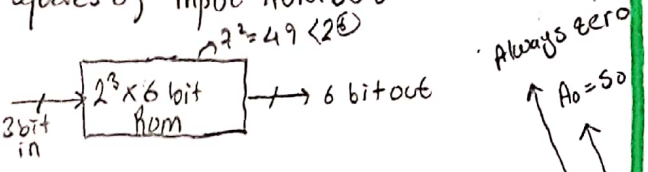
- fixed and array with 2^K outputs implementing all K literal minterms.
- Progo or array with n outputs lines to form up to n sum of minterm expressions.

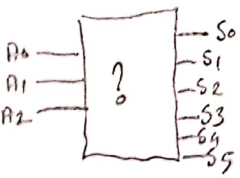
ROM Example



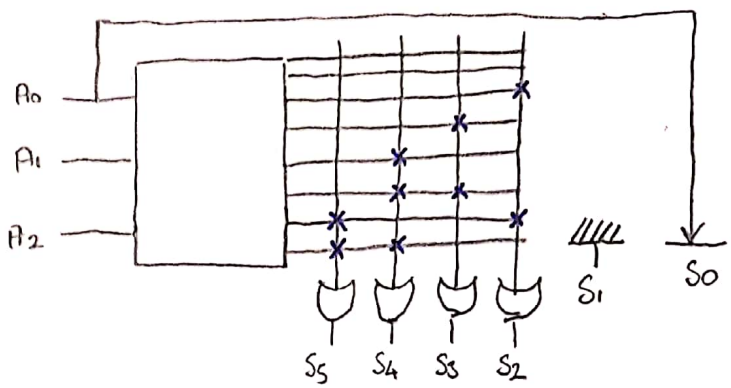
ROM Example

- implementing a combinational Logic Circuit using ROM. The circuit accepts a 3 bit number and generates an output binary number equal to the squares of input number.

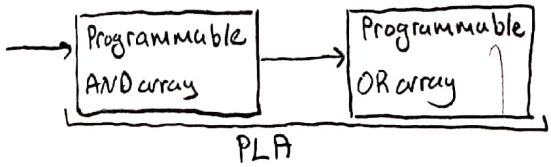




A_2	A_1	A_0	S_5	S_4	S_3	S_2	S_1	S_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1



PLA (Programmable Logic Array):



Compared to ROM, PLA is the most flexible having a programmable set of ands combined with a programable set of ORs.

Advantages:

- PLA can have large k and m permitting implementation of equations that are impractical for a ROM
- PLA has all of its product terms connectable to all outputs, overcoming the problem of the limited inputs to the PAL ORs.
- Some PLAs have outputs that can be complemented.

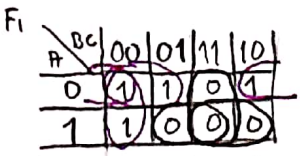
Disadvantages:

- Often the product terms count limits the application of PLA.
- Two level multiple output optimization is required to reduce the number of product terms in an implementation.

PLA Example

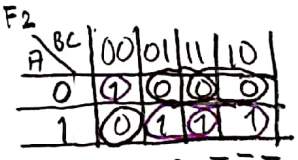
- implement the following two boolean functions with a PLA.

$F_1 = \sum_m(0,1,2,4)$, $F_2 = \sum_m(0,5,6,7)$ for (A,B,C)



$F_1 = \bar{A}\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{C}$

$\bar{F_1} = BC + AC + AB$



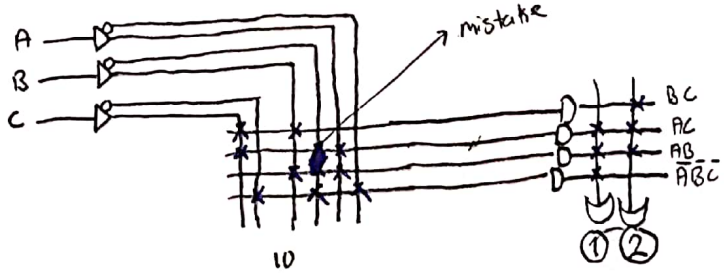
$F_2 = AC + AB + \bar{A}\bar{B}\bar{C}$

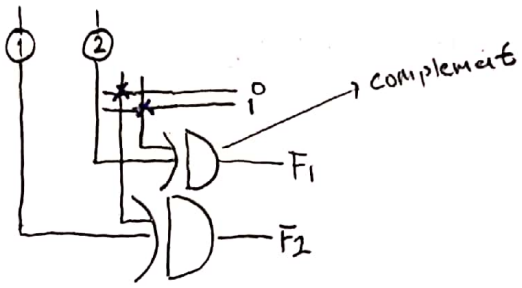
$\bar{F_2} = \bar{A}C + \bar{A}\bar{B} + A\bar{B}C$

Most common elements

PLA table

Product Term	INPUTS A	B	C	Outputs F_1^C	F_2^T
BC	-	1	1	1	-
AC	1	-	1	1	1
AB	1	1	-	1	1
$\bar{A}\bar{B}\bar{C}$	0	0	0	-	1





PAL (Programmable Array Logic):

PAL is opposite of ROM.

Advantages:

- For given internal complexity, PAL can have larger K and n.
- Some PAL's have outputs that can be complemented.

Disadvantages:

- ROM guaranteed to implement any n functions of K inputs. PAL may have too few inputs to the or gates.

PAL Example

$W(A,B,C,D) = \sum_m (2,12,13)$

$X(A,B,C,D) = \sum_m (7,8,9,10,11,12,13,14,15)$

$y(A,B,C,D) = \sum_m (0,2,3,4,5,6,7,8,10,11,15)$

$Z(A,B,C,D) = \sum_m (1,2,8,12,13)$

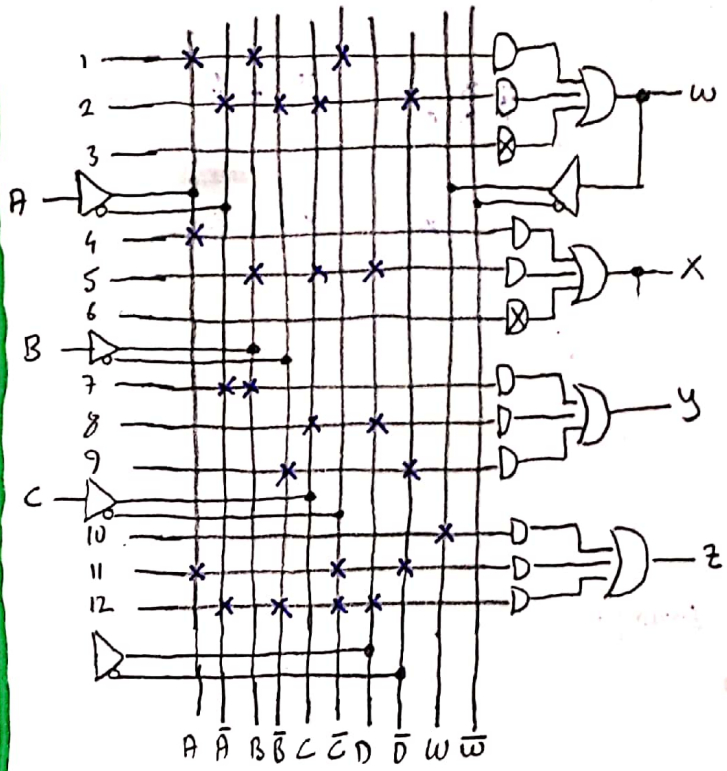
Simplifying after Karnaugh Maps...

$W = A\bar{B}\bar{C} + \bar{A}\bar{B}CD$ (terms 1, 2)

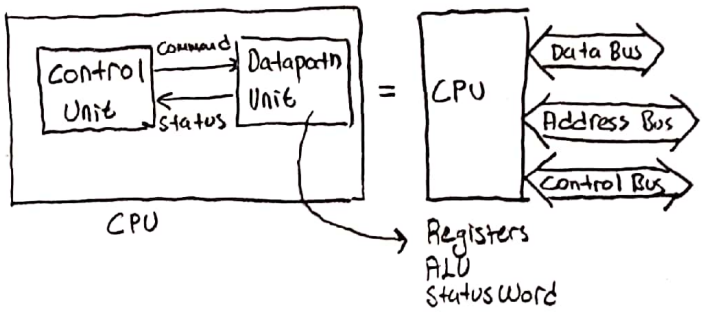
$X = A + BCD$ (terms 3, 4)

$y = \bar{A}B + CD + \bar{B}\bar{D}$ (terms 5, 6, 7)

$Z = A\bar{B}\bar{C} + \bar{A}\bar{B}C\bar{D} + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D = W + A\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D$ (terms 8, 9, 10, 11)



Computer Design Basics



Arithmetic Logic Unit (ALU):

Decompose the ALU into:

- An arithmetic circuit
- A logic circuit
- A selector to pick between two circuits

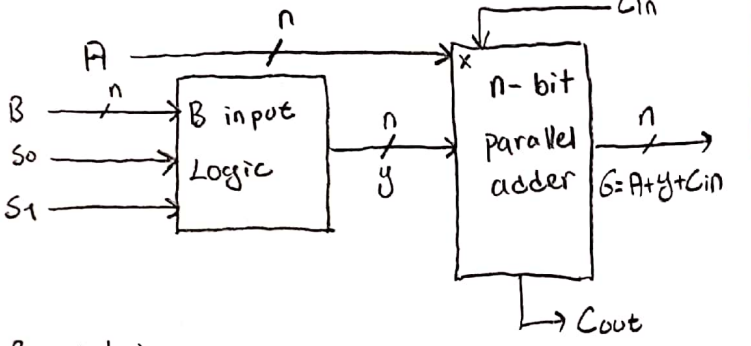
Arithmetic Circuit Design:

- An n-bit parallel adder
- A block of logic that selects four choices for the B input to the adder.

Arithmetic Circuit Design :

Select		Input	G=A+y+Cin	
S_1	S_0	y	Cin=0	Cin=1
0	0	all 0's	G=A	G=A+1
0	1	B	G=A+B	G=A+B+1
1	0	$\bar{B}$	G=A+$\bar{B}$	G=A+$\bar{B}$+1
1	1	all 1's	G=A-1 → twos complement	G=A

Subtract



B input logic

S_1	S_0	B_e	y_e
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

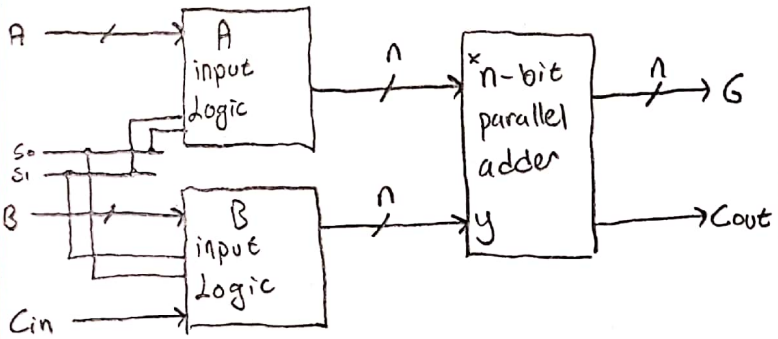
y_e

S_1 \ S_0B_e	00	01	11	10
0	0	0	1	0
1	1	0	1	1

$y_e = S_0 \cdot B_e + S_1 \cdot \bar{B_e}$

Arithmetic Circuit Design Example

S_1	S_0	Cin=0	Cin=1
0	0	G=A+B	G=A+$\bar{B}$+1
0	1	G=$\bar{A}$+B	G=$\bar{A}$+B+1
1	0	G=A-1	G=A+1
1	1	G=$\bar{A}$	G=$\bar{A}$+1



S_1	S_0	A_e	X_e
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

→ $X = S_0 \oplus A$

Cin	S_1	S_0	B_e	y_e	
0	0	0	0	0	
0	0	0	1	1	
0	0	1	0	0	
0	0	1	1	1	
0	1	0	0	1	-1
0	1	0	1	1	
0	1	1	0	0	yolk
0	1	1	1	0	
1	0	0	0	1	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	0	yolk
1	1	1	0	0	
1	1	1	1	0	

↳ $y = \bar{C_{in}}\,\bar{S_1}\,B + \bar{C_{in}}\,S_1\,\bar{S_0} + \bar{S_1}\,S_0\,B + C_{in}\,\bar{S_1}\,\bar{S_0}\,\bar{B}$

Arithmetic Circuit Design

S_1	S_0	Cin=0	Cin=1
0	0	G=A+B	G=A+B+1
0	1	G=$\bar{A}$	G=$\bar{A}$+1 → A+1-1
1	0	G=$\bar{B}$	G=A
1	1	G=A+$\bar{B}$	G=A+$\bar{B}$+1

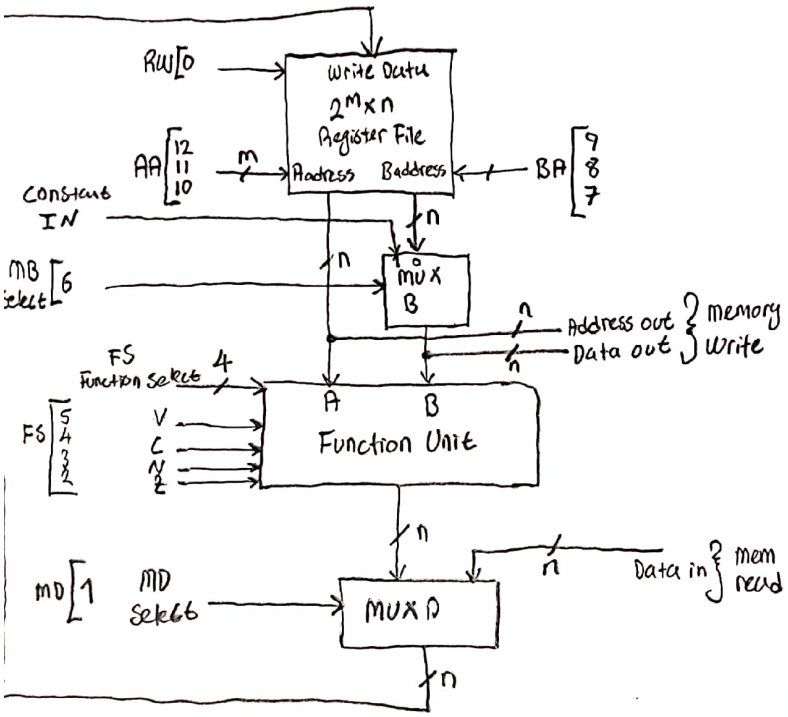
↳

S_1	S_0	Cin=0 X	Cin=0 Y	Cin=1 X	Cin=1 Y
0	0	A	B	A	B
0	1	A	0	A	0
1	0	0	$\bar{B}$	A	all 1s
1	1	A	$\bar{B}$	A	$\bar{B}$

Cin	S_1	S_0	X
0	0	0	A
0	0	1	A
0	1	0	0
0	1	1	A
1	0	0	A
1	0	1	A
1	1	0	A
1	1	1	A

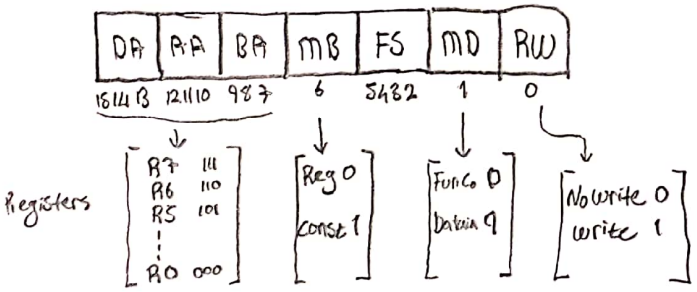
→ Karnaugh → $X = S_1A + S_0A + C_{in}A$

Instruction Example



Control Word Encoding

M = 3



Function Select

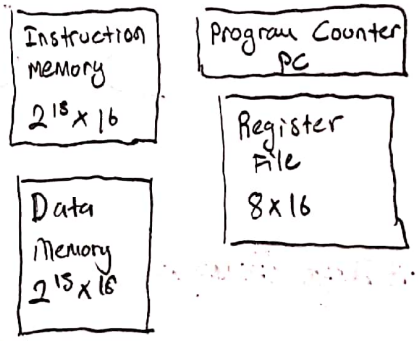
F ← A	0000	F ← A∧B	1000
F ← A+1	0001	F ← A∨B	1001
F ← A+B	0010	F ← A⊕B	1010
F ← A+B+1	0011	F ← $\bar{A}$	1011
F ← A+$\bar{B}$+1	0101	F ← B	1100
F ← A−1	0110	F ← srB	1101
F ← A	0111	F ← $\bar{B}$	1110

Microoperations

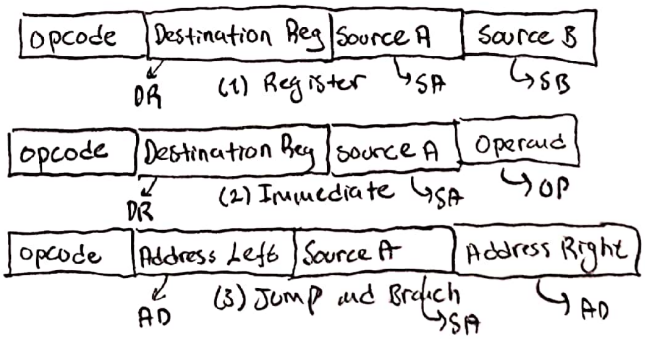
	DA	AA	BA	MB	FS	MD	RW
	R1	R2	R3	Reg	A+B+1	F	wr
$R_1 \leftarrow R_2 - R_3$	001	010	011	0	0101	0	1
	R4		R6	Reg	se	F	wr
$R_4 \leftarrow SCR6$	100	–	110	0	1110	0	1
	R7	R7			inc	F	wr
$R_7 \leftarrow R_7 + 1$	111	111	–	–	0001	0	1
	R1	R0		const	A+B	F	wr
$R_1 \leftarrow R_0 + 2$	001	000	–	1	0010	0	1
			R3	Reg			Nwr
Data out ← R_3	–	–	011	0	–	–	0
	R4					DI	wr
$R_4 \leftarrow$ Data in	100	–	–	–	–	1	1
	R5	R5	R5	Reg	A⊕B	F	wr
$R_5 \leftarrow \emptyset$	101	101	101	0	1010	0	1

Instructor Set Architecture (ISA) for Simple Computer (SC)

- A programmable system uses a sequence of instructions to control its operation.
- An typical instruction specifies:
  - Operation to be performed
  - Operands to use, and
  - Where to place the result, or
  - Which instruction to execute next.
- Instructions are stored in RAM or ROM.
- The addresses for instructions in a computer are provided by a program counter (PC) that can:
  - count up
  - Load a new address based on an instruction and, optionally status information.
- The PC and associated control unit are part of the Control Unit.



- A instruction consists of a bit vector.
- The fields of an instruction are subvectors representing specific functions.
- The format of an instruction defines the subvector and their functions.
- The SC ISA contains three format:



Example for 1: $R_1 \leftarrow R2 + R3$, $R1 \leftarrow \bar{6}R2$

Example for 2: $R_1 \leftarrow R_2 + 3$

Example for 3: if R6 == 0:

PC ← PC − 20

Control Unit:

- The data memory has been attached to the Address Out and Data Out and Data In lines of the Datapath.
- The MW input to the Data memory is the Memory Write signal from the Control Unit.
- The Instruction Memory address input is provided by the PC and its instruction output feeds the Instruction Decoder.
- The PC is controlled by Branch Control logic.

PC Function:

- The PC function is based on instruction specifications involving jumps and branches

- Branch on Zero (BRZ): if(R[SA]==0) PC ← PC+sEAD
- Branch on Negative (BRN): if(R[SA]<0) PC ← PC+sEAD
- Jump (JMP): PC ← R[SA]

- The Branch Control determines the PC transfers based on five of its inputs defined as follows
  - N, Z → Negative and zero status bit
  - PL → load enable for PC
  - JB → Jump/Branch Select | Jump=1, Branch=0
  - BC → Branch Condition Select: If BC=1, branch for N=1, else Branch for Z=1.

PC Operation	PL	JB	BC
Count UP	0	X	X
Jump	1	1	X
Branch on Negative	1	0	1
Branch on Zero	1	0	0

Instruction Example

	DA	AA	BA	MB	FS	MD	RW	MW	PL	JB
R[0] := R[3] ⊕ R[3]	000	111	011	0	1010	1	1	0	0	1
R[1] ← M[R[4]]	001	-	100	0	0000	1	1	0	0	1
R[2] ← R[3]+2	010	101	-	0	0010	0	1	0	0	1
R[3] ← SIR[6]	011	-	110	0	1110	0	1	0	0	1
if(R[4]=0): PC ← PC+sePC else: PC ← PC+1	-	-	100	-	-	-	0	0	1	0

Single-Cycle Computer Issues

- Complexity of instructions executable in a single cycle is limited.
- Accessing both an instruction and data from a single memory is impossible.
- A long worst case delay path limits clock frequency and the rate of performing instructions.

How to handle?

- The first two issue can be handled by the multiple-cycle computer.
- The third issue is dealt with by pipelining.

Multiple-Cycle Computers

Data path modifications:

- Uses a single memory for both instructions and data.
- Requires new MUX M1 with control signal MD1 to select the instruction address from the PC.
- Register file becomes 16x16
- Addresses to Register File increase from 3 to 4.
- Three new control fields for register address source selection and temporary storage addressing: AX, BX, DX

Control unit modifications:

- Requires an Instruction Register (IR) to hold the instruction
- Requires the addition of a "hold" operation to the PC since it only counts up to obtain a new instruction.

Encoding For Datapath Control

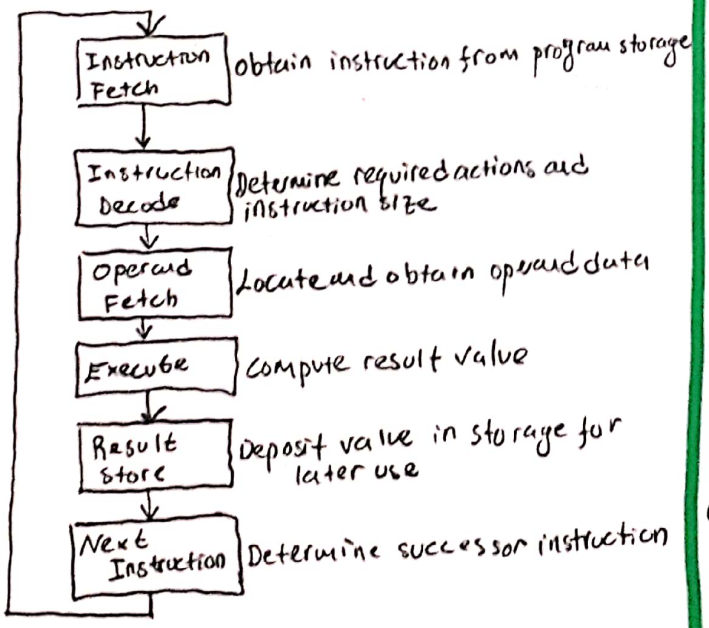
	RW	MD	FS	MB	BX	AX	DX
R9 ← R8+R15	1	0	0010	0	1111	1000	1001
R11 ← R12+1	1	0	0001	0	-	1100	1011

Instruction Set Architecture

Example ISAs:

- RISC (Reduced Instruction Set Computer)
  - IBM PowerPC
  - Sun Sparc etc
- CISC (Complex Instruction Set Computer)
  - Intel x86
  - Motorola 68000
- VLIW (Very Large Instruction Word)
  - Intel Itanium

- A processor is specified completely by its instruction set architecture.
- Each ISA will have a variety of instructions and instruction formats, which will be interpreted by the processor's control unit and executed in the processor's datapath.



Register Set:

- Programmer accessible registers (R0 to R7 in multi-cycle computer)
- Registers in the register file accessible only to microprograms (R8 to R17)
- Instruction register (IR)
- Program Counter (PC)
- Pipeline register
- Stack pointer (SP)

Operand Addressing:

Operand: register value, memory content, or immediate

Explicit Address: address field in the instruction

Implied Address: The location of operand is specified by the opcode or other operand address.

Three Address Instructions

Example: $X = (A+B)(C+D)$

ADD T1, A, B	$M[T1] \leftarrow M[A]+M[B]$
ADD T2, C, D	$M[T2] \leftarrow M[C]+M[D]$
MUX X, T1, T2	$M[X] \leftarrow M[T1] \times M[T2]$

OR

ADD R1, A, B	$R1 \leftarrow M[A]+M[B]$
ADD R2, C, D	$R2 \leftarrow M[C]+M[D]$
MUX X, R1, R2	$X \leftarrow R1 \times R2$

+: short program, 3 instructions

−: binary coded instruction requires more bits to specify three address.

Two Address Instruction

Example: $X = (A+B)(C+D)$

```
MOVE T1, A
ADD  T1, B
MOVE X, C
ADD  X, D
MUL  X, T1
```

5 instructions

One Address Instructions

Implied address: a register called an accumulator ACC for one operand and the result, single accumulator architecture.

LD	A	$ACC \leftarrow M[A]$
ADD	B	$ACC \leftarrow ACC+M[B]$
ST	X	$M[X] \leftarrow ACC$
LD	C	$ACC \leftarrow M[C]$
ADD	D	$ACC \leftarrow ACC+M[D]$
MUX	X	$ACC \leftarrow ACC \times M[X]$
ST	X	$M[X] \leftarrow ACC$

7 instructions

All operations between the ACC and memory operand

Zero Address Instructions

-Use stack

```
PUSH A
PUSH B
ADD
PUSH C
PUSH D
ADD
MUL
POP X
```

- Data manipulation operations: between the stack elements
- Transfer operations: between the stack and memory

Addressing Architecture:

Two kinds of addressing architectures:

- memory-to-memory
  - only one register - PC
  - All operands from memory, and results to memory
- register-to-register
  - restrict only one memory address to load/store types, all operations are between registers.

Addressing Modes:

- Implied mode: implied in opcode, such as stack and ACC.
- Immediate mode (operand): a = 0x08 01 234
- Register mode: a = R[b]
- Register-Indirect mode: a = M[R[b]]
- Direct adressing mode: a = M[0x0018df8]
- Indirect adressing mode: a = M[M[0x0018df8]]
- PC-relative adressing: offset + PC
- Indexed adressing: a = b[1]

	RISC	CISC
Memory Access	restricted to load/store instructions, and data manipulation instructions are register-to-register	is directly available to many types of instructions.
Addressing mode	limited in number	substantial in number
Instruction format	all of the same length	of different lengths
Instructions	perform elementary operations	perform both elementary and complex
Control unit	Hardwired, high throughput and fast execution	microprogrammed, facilitate compact programs and conserve memory.

Conditional Branching:

Branch Condition	Mnemonic	Tests
Branch if zero	BZ	Z=1
Branch if not zero	BNZ	Z=0
Branch if carry	BC	C=1
Branch if no carry	BNC	C=0
Branch if minus	BN	N=1
Branch if plus	BNN	N=0
Branch if overflow	BV	V=1
Branch if no overflow	BNV	V=0

- For Unsigned Numbers

Branch Condition	Mnemonic	Condition	Status Bit
Branch if above	BA	A>B	C+Z=0
Branch if above or equal	BAE	A≥B	C=0
Branch if below	BB	A<B	C=1
Branch if below or equal	BBE	A≤B	C+Z=1
Branch if equal	BE	A=B	Z=1
Branch if not equal	BNE	A≠B	Z=0

C is borrow bit

- for Signed Numbers

Branch Condition	Mnemonic	Condition	Status Bits
Branch if greater	BG	A>B	(N⊕V)+Z=0
Branch if greater or equal	BGE	A≥B	(N⊕V)=0
Branch if less	BL	A<B	(N⊕V)=1
Branch if less or equal	BLE	A≤B	(N⊕V)+Z=1
Branch if equal	BE	A=B	Z=1
Branch if not equal	BNE	A≠B	Z=0

Interrupts:

External: Hard drive, mouse, keyboard, modem, printer

Internal: Overflow, divide by zero, invalid opcode, memory stack overflow, protection violation

Software: External or internal interrupts by inserting an instruction into the code.

Floating Point Computation:

Unsigned ⟶ 0 to 2^{n-1}

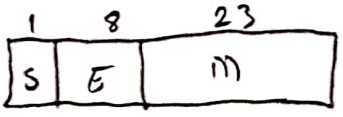
2s complement ⟶ -2^{n-1} to $2^{n-1}-1$

1s complement ⟶ $-2^{n-1}+1$ to $2^{n-1}-1$

BCD ⟶ 0 to $10^{n/4}-1$

Real Scientific Notation

IEEE FoP $\pm 1.M \times 2^{e-127}$

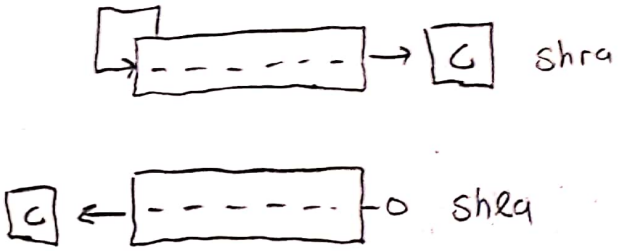


$N = (-1)^S \cdot 2^{E-127} (1.M)$

Example 1

Convert the following decimal numbers to signed binary. Perform the the arithmetic shift left and right operations and indicate the overflow bit.

(+62)	(-75)
$(+62)_{10} = 00111110$	$(75)_2 = 01001011$ ⇩ 10110100 $+ \quad 1$ $(-75)_2 = 10110101$
Shla → 01111100, V=0 Shra → 00011111, → V=0 → No overflow	shla → 01101010 V=0 Shlr → 11011010 ↳ no overflow



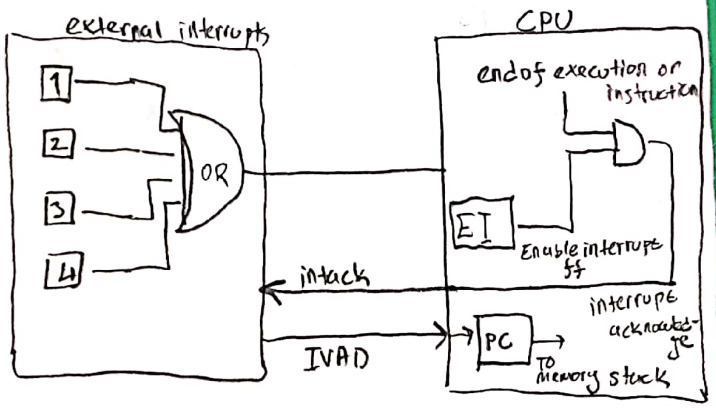
Example 2

- The program in a computer compares two signed 2's complement number A and B by performing subtraction A-B and updating status bits.
  - Determine N, Z, V

Mnemonic	Condition	Test	A=01010010 B=11012001	
A-B			10000001	
N, Z, V		V=0, Z=0, N=1		V=1, Z=0, N=1
BG	A>B	(N⊕V)+Z=0	False (1⊕0)+0=1	True (1⊕1)+0=0
BGE	A≥B	(N⊕V)=0	False (1⊕0)=1	True (1⊕1)=0
BL	A<B	(N⊕V)=1	True (1⊕0)=1	False (1⊕1)=0
BLE	A≤B	(N⊕V)+Z=1	True (1⊕0)+Z=1	False (1⊕1)+0=0

Processing External Interrupts:

SP ← SP-1	Decrement stack pointer
M[SP] ← PC	Store return address on stack
SP ← SP-1	Decrement stack pointer
M[SP] ← PSR	Store processor status word on stack
EI ← 0	Reset enable interrupt flip flop
INTACK ← 1	Enable interrupt acknowledge
PC ← IVAD	Transfer interrupt vector address to PC and GO to fetch.



Input - Output

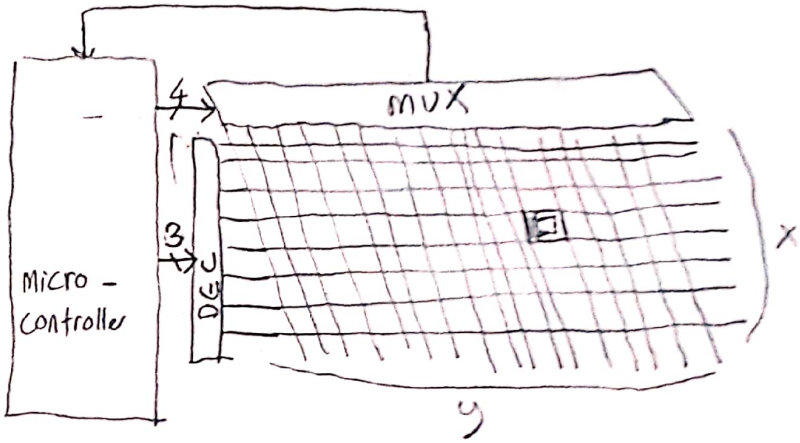
- keyboards
- displays
- printers
- magnetic devices
- compact disc read only memory
- modems, scanners, speakers, mics etc

Sample - Peripherals:

- Devices that CPU controls directly are said to be connected online
- Input or outputs devices attached to the computer online are called peripherals.
  - Keyboard
  - Hard drive
  - Liquid Crystal Display Screen

Keyboard:

- Scan matrix lies beneath the keys
- Whether a key depressed and released, the control code at the time of the event is sensed and is translated by the microcontrolled into K-scan code.
- When a key depressed, a make code is produced, when a key is released a break code is produced.

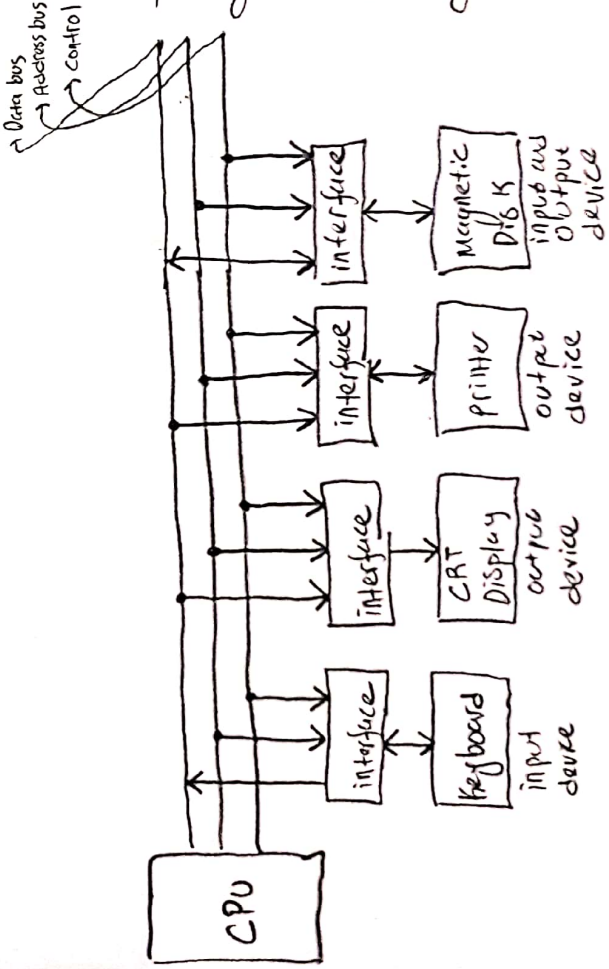


Hard Drive:

- The hard drive is the primary intermediate speed nonvolatile, writable storage medium.
- Revolutions per Minutes (RPM): rational speed of a disk
- Disk access time and transfer time

LCD Screen:

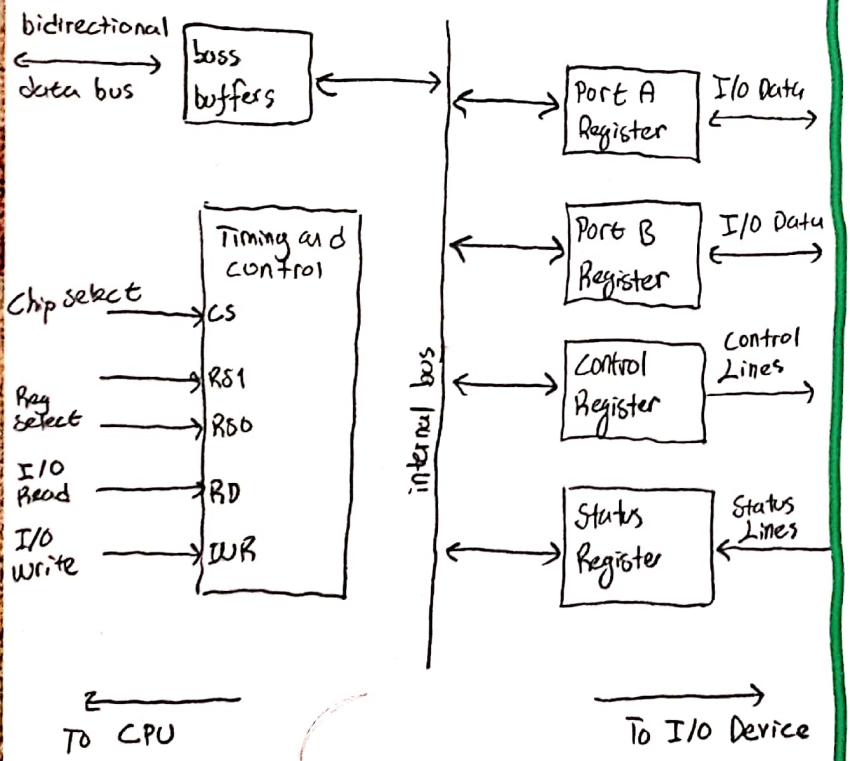
- The display screen is defined in terms of picture element called pixels.
- The color display has three subpixels corresponding to the primary colors; red, green and blue (RGB)



I/O Configurations:

- Memory-mapped: Use common data, address, and control busses for both memory and I/O.
- Isolated: Share a common address and data bus but use different control lines (eg intel 8086)
  - IN, OUT instructions for Access I/O ports
  - MOVE instruction to access memory

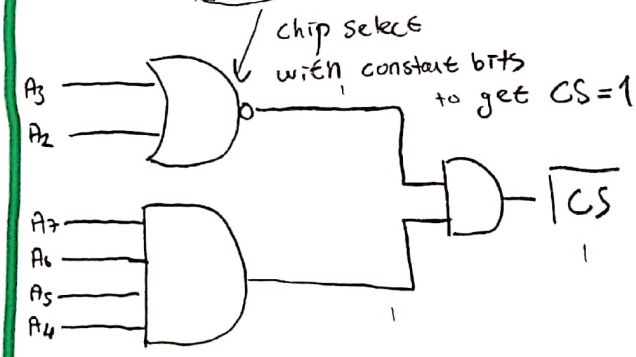
Example of I/O Interface



CS	RS1	RS0	Register Selected
0	X	X	None
1	0	0	Port A
1	0	1	Port B
1	1	0	Control Reg
1	1	1	Status Reg

- The address assigned to the four registers of the I/O interface on previous figure are equal to the binary equivalat of 240, 241, 242 and 243. Show the external circuit that must be connected between an 8-bit I/O address from the CPU and CS, RS0, RS1 inputs of the interface.

I/O Addres Decimal	I/O Adress Binary	Register selected	Control Inputs RS1=A1 and RS0=A0
240	11110000	Port A	RS0=0 RS1=0
241	11110001	Port B	RS0=1 RS1=0
242	11110010	control	RS0=0 RS1=1
243	11110011	Status	RS0=1 RS1=1



Example of I/O Interface

- **Example:** Six interface units of the type shown in previous slide are connected to a CPU that uses an I/O address of eight bits. Each one of the six chip select (CS) inputs is connected to a different address line. Specifically, address line 0 is connected to the CS input of the first interface unit, and address line 5 is connected to the chip select input of the sixth interface unit.
- Address lines 7 and 6 are connected to the RS1 and RS0 inputs, respectively, of all six interface units. Determine the 8-bit address of each register in each interface (a total of 24 addresses)
- Solution ???

6 interface → A_0 to CS_1 A_5 to CS_6

8 bit → $A_7 = RS1$

$A_6 = RS0$

(1) I/O Add

Dec	I/O Add Bin	Reg
1	00000001	
65	01000001	
129	10000001	
193	11000001	

(2) I/O add

Dec	I/O Add Bin	Reg
2	00000010	
66	01000010	
130	10000010	
194	11000010	

(3) I/O add

Dec	I/O Add Bin	Reg
4	00000100	
68	01000100	
132	10000100	
196	11000100	

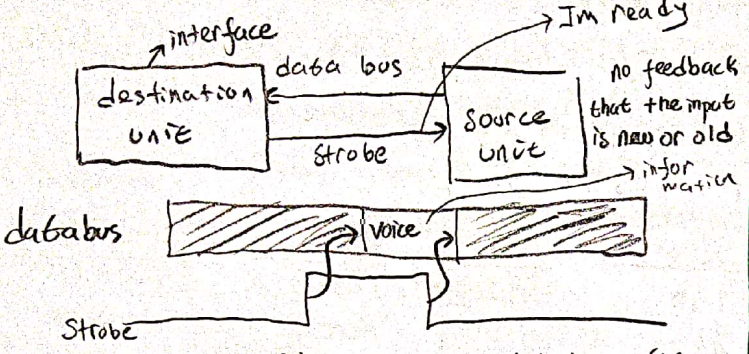
(4) I/O Add

Dec	I/O Add Bin	Reg
8	00001000	
72	01001000	
136	10001000	
200	11001000	

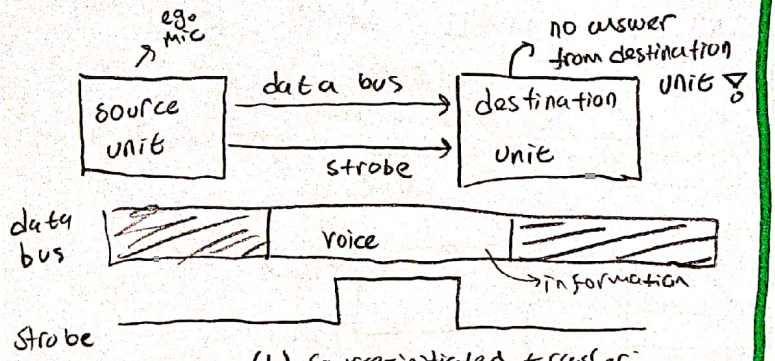
5 and 6 is same

Strobing:

The databus between two units is assumed to be made bidirectional by the use of three state buffers.

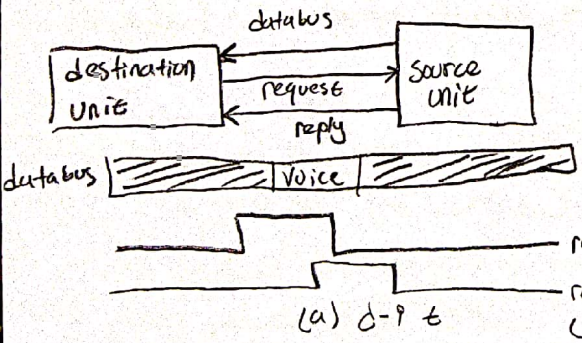


(a) destination-initiated transfer

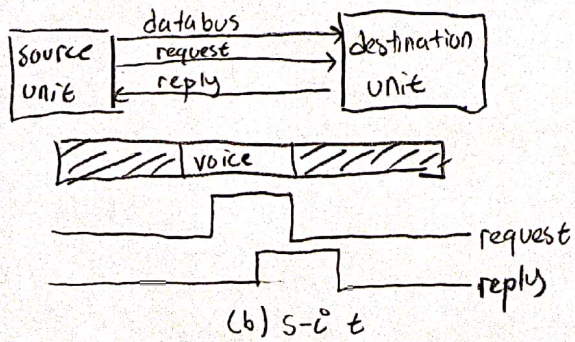


(b) Source-initiated transfer

Handshaking:



(a) 2-wire



(b) 3-wire

Serial Communication:

- Parallel Communication
  - transfers more than one bit of data at a given time
  - N-bits transmitted at the same time through N wires
  - faster but require many wires
- Serial Communication
  - cannot handle more than one bit
  - slower but less wires

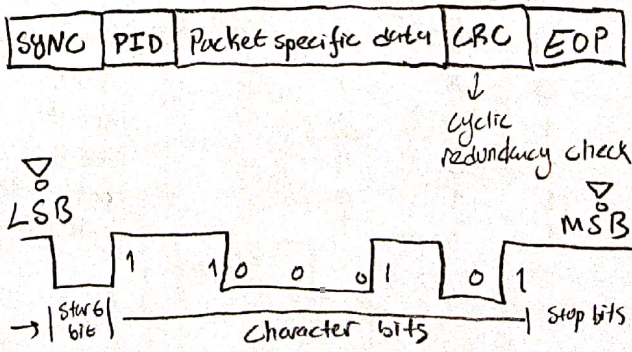
Asynchronous Serial Communication

- interacts with devices outside of the computer.
  - Ex: modems, printers
- Transmit individual bytes instead of large blocks
- Do not share a common clock.

(start bit)

Synchronous Serial Communication

- Transmits block of data in frames.
  - Frames are had head in front of the data and a tail at the end of the data.
- The head and tail contain information that allows the two computers to synchronize their clocks.



Device 1 start bit → Device 2 receives and confirms → Device 2 read bit by bit → Device 2 receives and confirms stop bit

Asynchronous Serial Communication Example

- How many characters per second (velocity) can be transmitted over a 57,600-bound line in each of the following modes? (Assume a character code of 8 bits)

a) Asynchronous serial transmission with odd-parity bit and two stop bit.

b) Draw a time diagram for sending character 'A'.

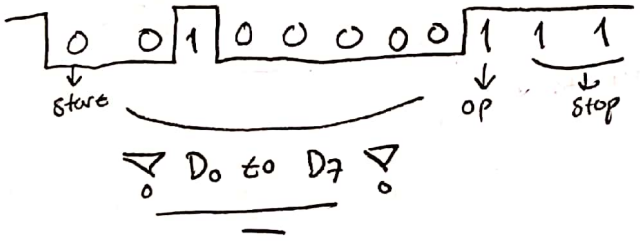
a') 1 start bit + 8 data bits + 1 parity bit + 2 stop bits = 12 bits

57,600 bps / 12 bits = 4,800 char per second.

b') ASCII of 'A' is $(65)_{10} = (01000001)_2$

0 | 1000 0010 | 1 | 11

0: start bit; 1000 0010: data code; 1: odd parity; 11: stop bits



Universal Asynchronous Receiver/Transmitters (UARTS)

It is used to convert serial communication when receiving and convert parallel communication to serial when sending.

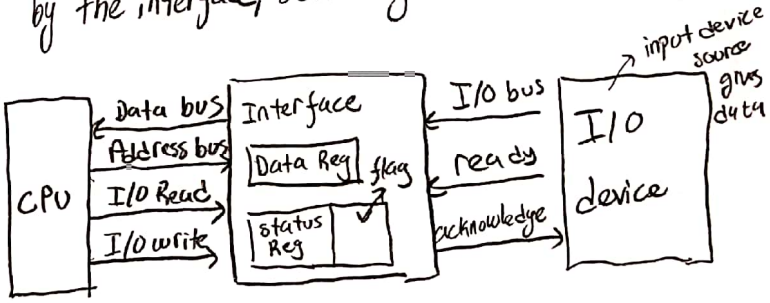
Modes of Transfer: (slow)

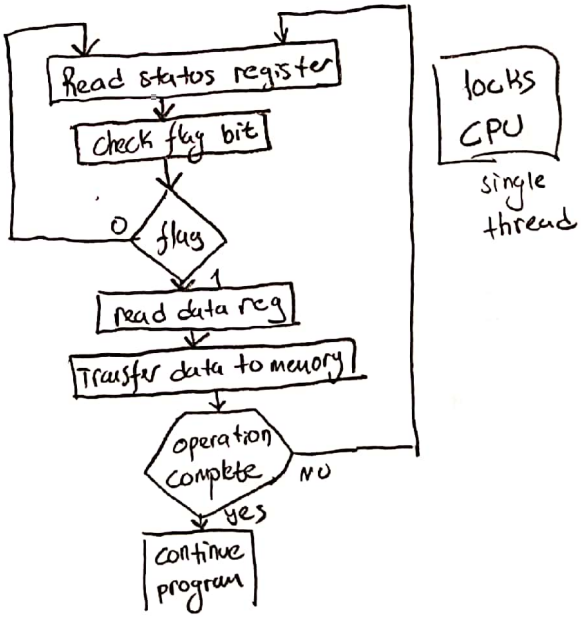
- Data transfer under program control.
- Interrupt-initiated data transfer.
- Direct memory access (DMA) transfer. (HDD?)
- Transfer through an I/O Processor. (mainframe)

1-Data Transfer Under Program Control

I/O to CPU

- The device transfers bytes of data one at a time as they are available. When available, the device places it on the I/O bus and sets Ready.
- The interface accepts the byte into its register and sets Acknowledge
- The interface sets a bit in the status register, which we will refer to as a flag. Device can now reset Ready, but it won't transfer another byte until Acknowledge is reset by the interface, according to the handshaking.

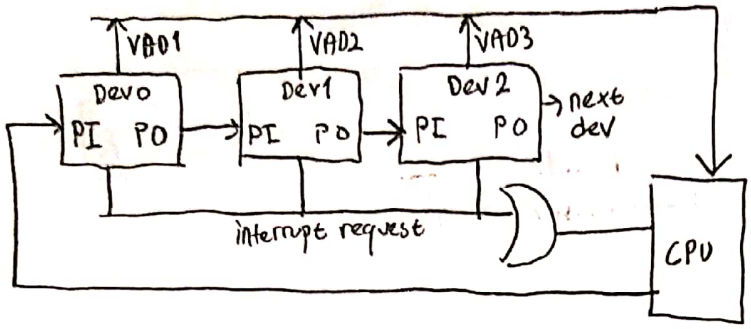


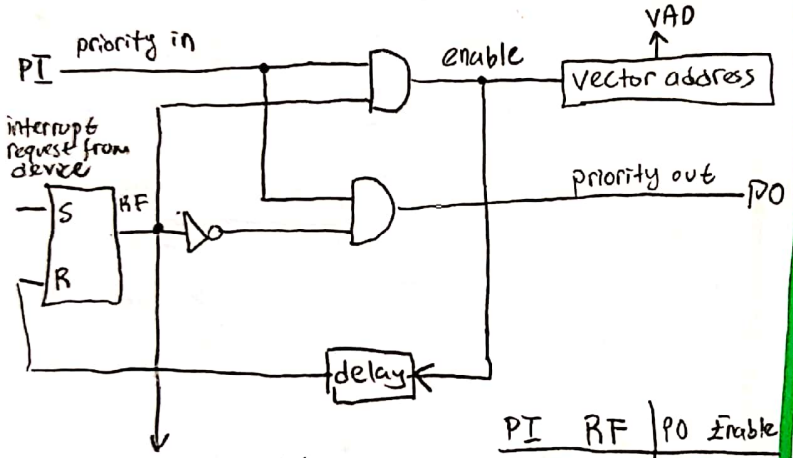


2- Interrupt Initiated Data Transfer

a) Daisy chain priority

process priority (device)





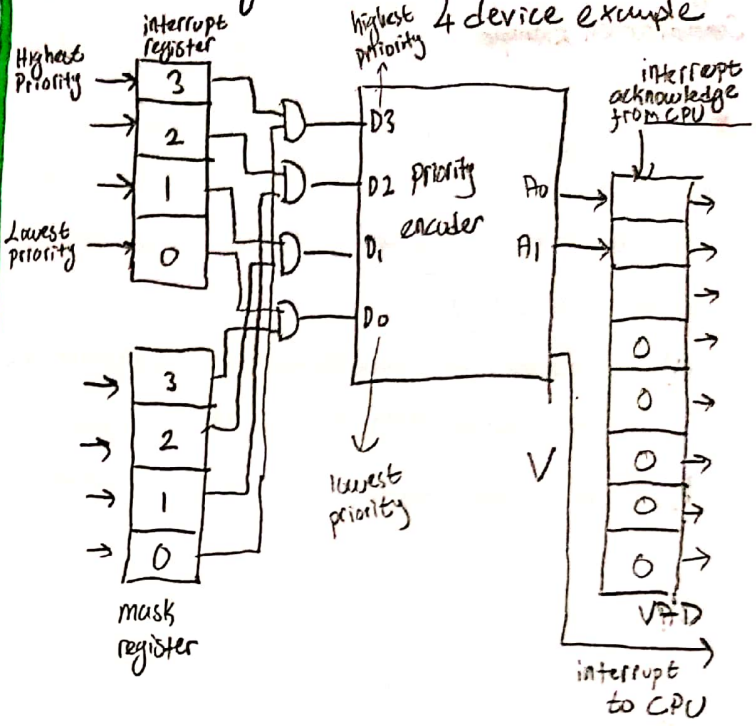
PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	0	1

if working speed ↑
then priority ↑
else priority ↓
fi

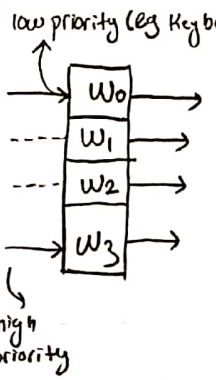
→ if request at same time

b) Parallel Priority Hardware

4 device example



Priority Encoder :



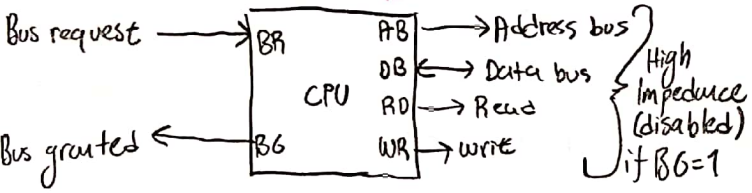
4 different I/O device

No IR

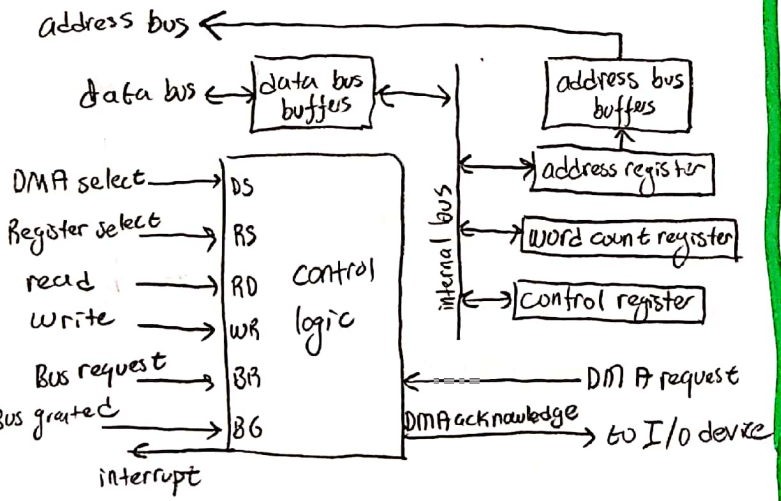
W_3	W_2	W_1	W_0	y_1	y_0	Z (valid)
0	0	0	0	–	–	0
0	0	0	1	0	0	1
0	0	1	–	0	1	1
0	1	–	–	1	0	1
1	–	–	–	1	1	1

3- Direct Memory Access

- DMA improves system performance by speeding up data transfer between memory and I/O system.
- Bypass CPU, allow CPU to be use in another process.
- DMA controllers must manipulates each data transfer from I/O devices, and only can read



DMA Controllers



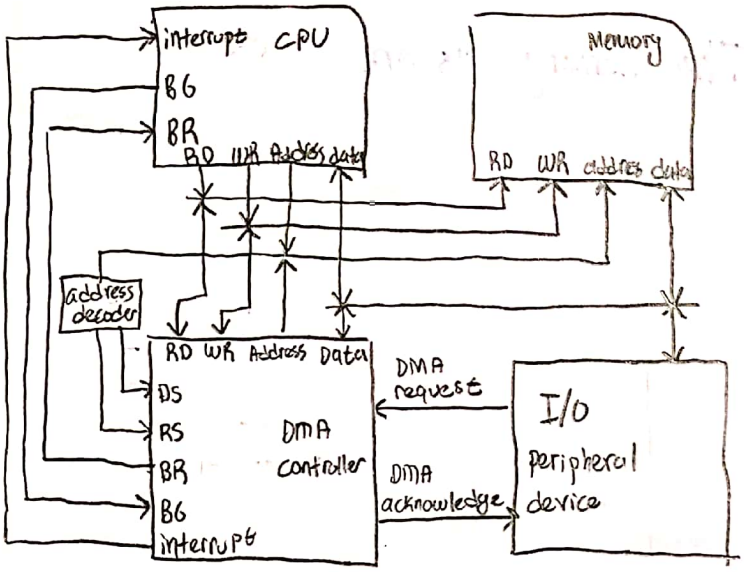
1- the starting address of the memory block in which data is available (for reading) or data is to be stored (for writing).

2- The word count, which is the number of words in the memory block.

3- A control bit to specify the mode of transfer, such as read or write.

4- A control bit to start the DMA transfer.

CPU is out of order when this transfer happens.



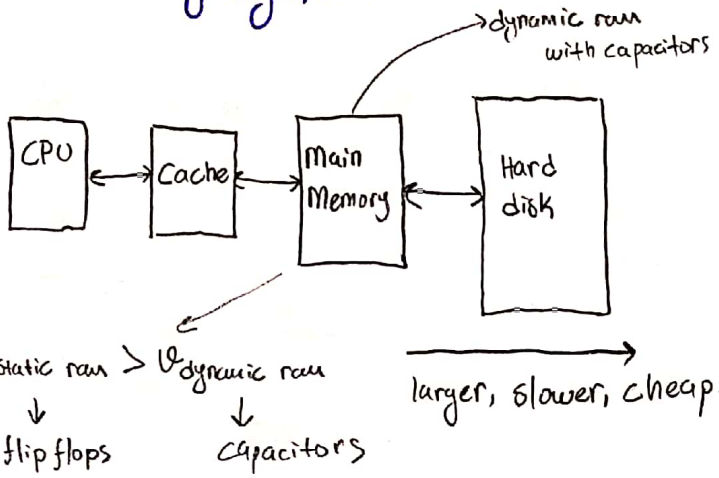
needs start control bit to transfer

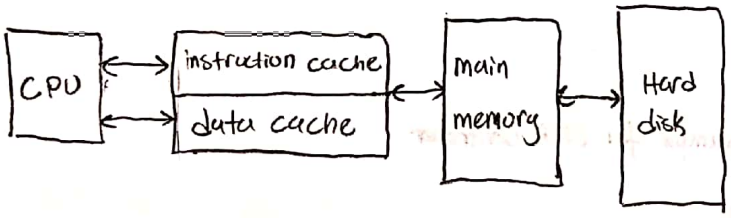
Example for DMA Controller

- Why are the read and write control lines in a DMA controller bidirectional?

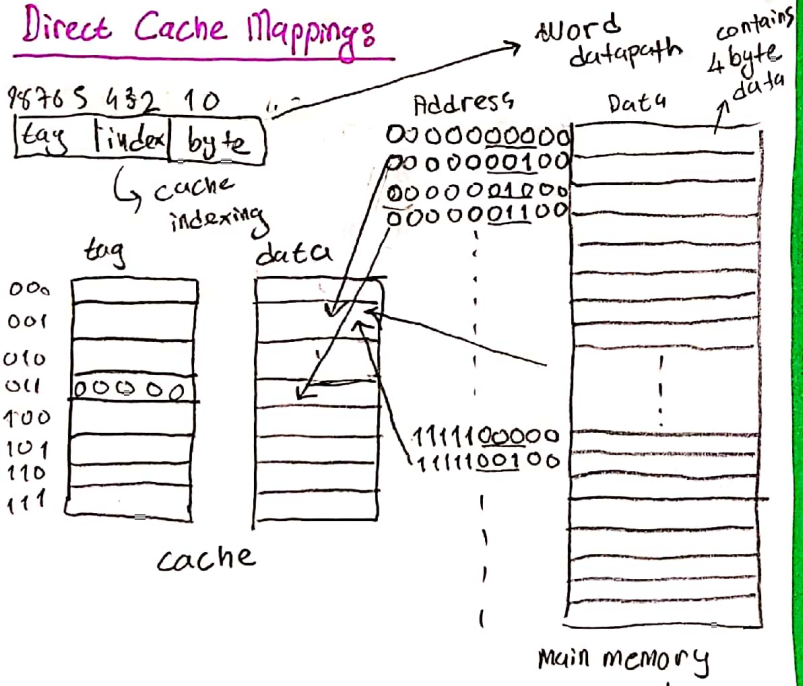
DMA controller needs to be configured before it is let loose to do the memory transfer. Registers in DMA controller need to be written to and read from, and hence for that purpose they are inputs, however when they are used by the DMA controller to do the DMA transfer, they are outputs.

The Memory System





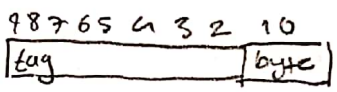
Direct Cache Mapping:

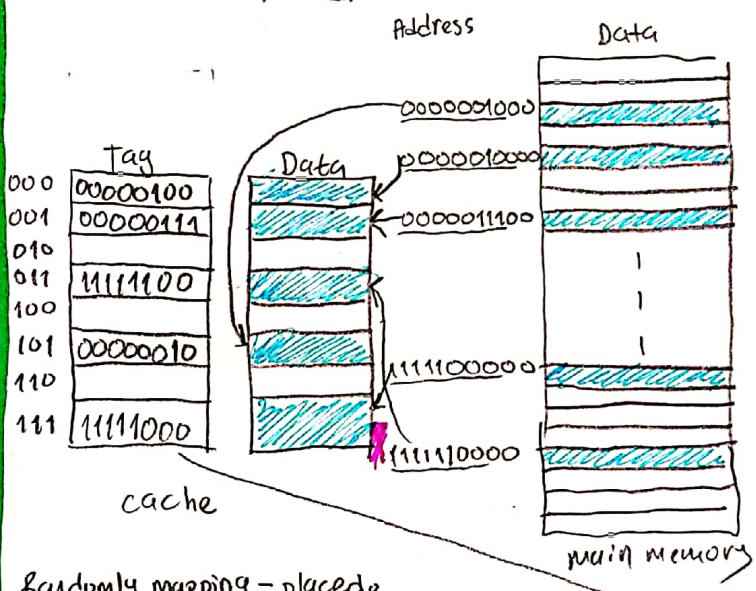


Maps each block of main memory into only one possible cache line. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block trashed.

2 matching logic

Fully Associative Mapping:





Randomly mapping - placed.

Cache hit → data found in cache

Cache miss → data not found in cache

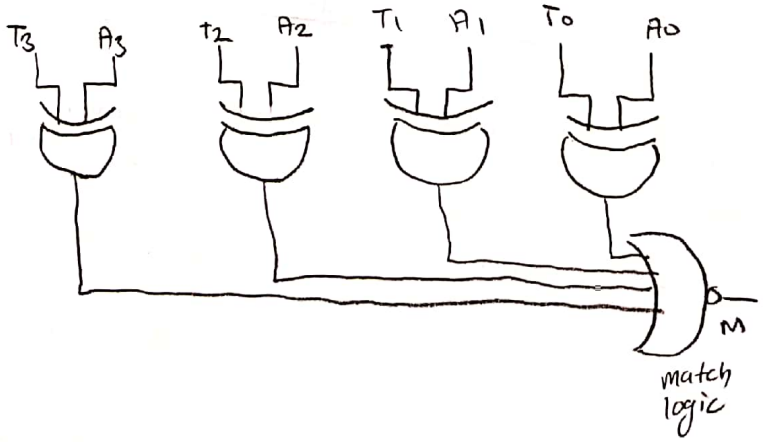
comparing what is needed at CPU and tag then matching.

In this case we need 8 comparing circuit.

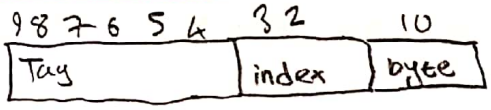
↳ n (row)$_{tag}$

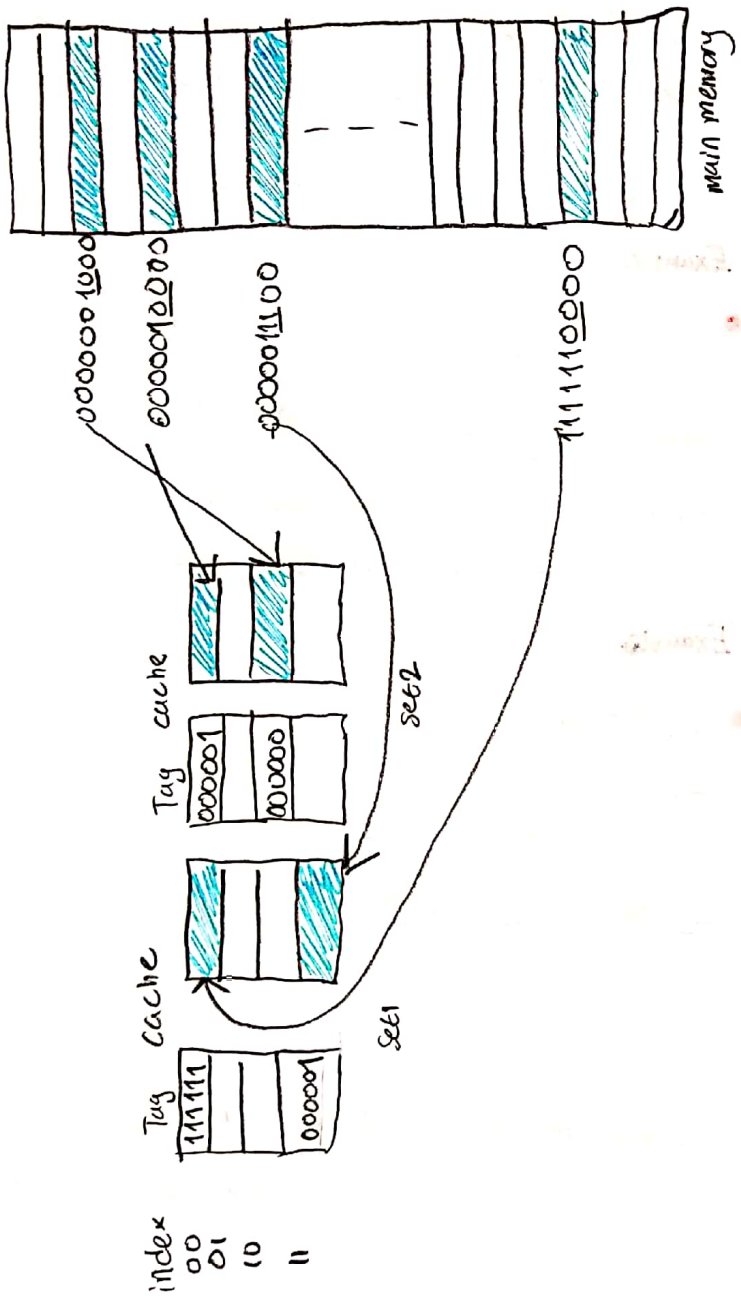
parallel comparing

So it is computationally expensive.



Set Associative Mapping:





common indexing. Capacity is 8 like fully associative but has two set. No need to overwrite

2 matching logic and 6 xor gate
↳ n (set)
↳ n (tag)

Principal of Locality:

- Cache work on a principal known as locality of reference. This principal states asserts that programs continually use and re-use the same locations.
  - instructions: Loops, common subroutines
  - Data: look-up tables, arrays
- Temporal Locality is same location will be refereced again soon.
- Spatial Locality is nearby locations will be refereced soon.

Direct Mapped Cache:

- in a direct-mapped cache, each memory address is associated with one possible block within the cache due to spatial locality.
  - We only need to look single location in cache for the data if exists.

Example for Direct-Map Cache:

- Suppose we have a 16KB of data in a direct mapped cache with 4 word blocks. Determine size of the tag, index and offset fields if we are using a 32-bit architecture. ↳ 1 word
  - offset needs to specify correct byte within a block
    block contains 4 word = 16 byte = 2^4 bytes
    needs 4 bits to specify correct byte
    ↳ 2 bit byte offset
    ↳ 2 bit word offset
  - index needs to specify correct row in cache
    cache contains 16KB = 2^{14} bytes
    block contains 24 bytes (4 words)
    $2^{14}/2^4 = 2^{10}$
    needs 10 bits to specify this many rows
  - Tag: 32 - 4 - 10 = 18 bits

address (hex)	value of word
00000010	a
00000014	b
00000018	c
0000001C	d
...	
00000080	e
00000034	f
00000038	g
0000003C	h
...	
00008010	e
00008014	i
00008018	k
0000801C	l

read 4 addresses

- 0x00000014
- 0x0000001C
- 0x00000034
- 0x00008014

↓

tag	index	offset
00000000000000000	0000000001	0100
00000000000000000	0000000001	1100
00000000000000000	0000000011	0100
00000000000000010	0000000001	0100

cache

index	Tag	data 0x0-3	0x4-7	0x8-b	0xc-f
0					
1	0	a	b	c	d
2					
3	0	e	f	g	h

Average Access Time:

Average access time, t_a, given by:

$$t_a = t_c + (1-h)\,t_M$$

Example

- if hit ratio is 0.85, main memory access time is 50 ns and cache access time is 5ns, what will be average access time?

$$5 + (1-0.85)*50 = 12.5\ ns$$

Example

- hit time = 1cycle → cache
  miss rate = 5%
  miss penalty = 20 cycles → main memory

  Average memory access time is

  $$1 + 0.05 \times 20 = 2\ cycle$$

Example

- L_1 Hit time = 1cycle
  L_1 miss rate = 5%
  L_2 Hit time = 5cycle
  L_2 miss rate = 15%
  L_2 miss penalty = 100 cycles

→ L_1 miss penalty = $5 + 0.15 * 100 = 20$

→ Average memory access = $1 + 0.05 \times 20 = 2$ cycle