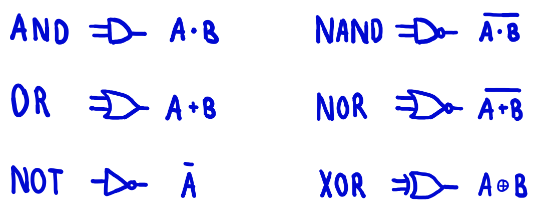
CS210 Cheatsheet

Logic Gates



AND	$A \cdot B$	NAND	$\overline{A \cdot B}$
OR	$A + B$	NOR	$\overline{A + B}$
NOT	$\bar{A}$	XOR	$A \oplus B$

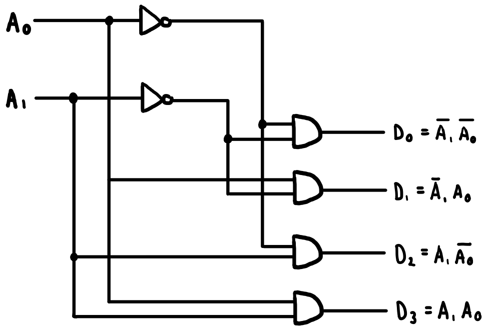
XOR TRUTH TABLE

A	B	RESULT
0	0	0
0	1	1
1	0	1
1	1	0

Definitions

Decoders	Recognizes specific bit patterns
Multiplexors	chooses among various inputs

Example of a decoder

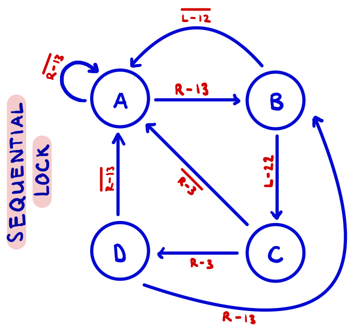


TRUTH TABLE

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

State Diagrams

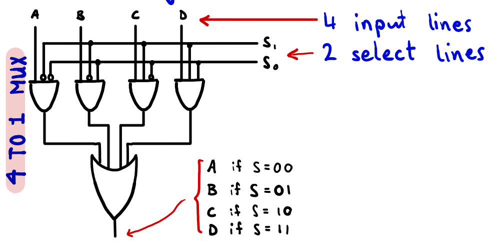
→ Show states and actions that cause transition between states



Multiplexer (Mux)

→ A mux has 2^n data inputs, n select lines, and 1 output

→ The select bits are to "choose" one of the data inputs to flow through to the output

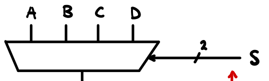


4 input lines

2 select lines

A if S=00
B if S=01
C if S=10
D if S=11

STANDARD SYMBOL FOR MUX



Indicates a 2-bit signal

Instruction Processing

Fetch

1) loads next instruction from memory store at address in PC and places into Instruction Register (IR)

2) PC is increment to point to next instruction

Decode

1) Identifies the opcode

2) Depending on opcode, identifies other operands from remaining bits

Evaluate Address - For operations that require memory access, compute address for access

Fetch Operands - Obtain source operands needed to perform operation

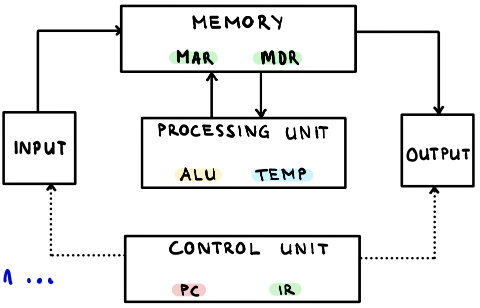
Execute - Perform the operation, using source operands

Store - Write results to destination (register or memory)

Fetch Instruction (from memory)	F
↓ Decode Instruction	D
↓ Evaluate Address	EA
↓ Fetch Operands	OP
↓ Execute Operation	EX
↓ Store Result	S

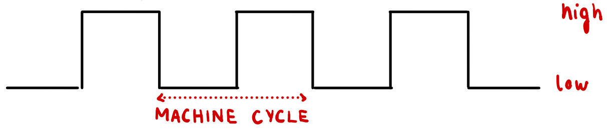
LC3 Written as Von Neumann Model

MAR	Memory Address Register	PC	Program Counter
MDR	Memory Data Register	ALU	Arithmetic Logic Unit
IR	Instruction Register	INPUT	Keyboard/Mouse/...
		OUTPUT	Screen ...



The Clock is a signal which keeps the Control Unit moving. Each clock tick moves the CU to the next step.

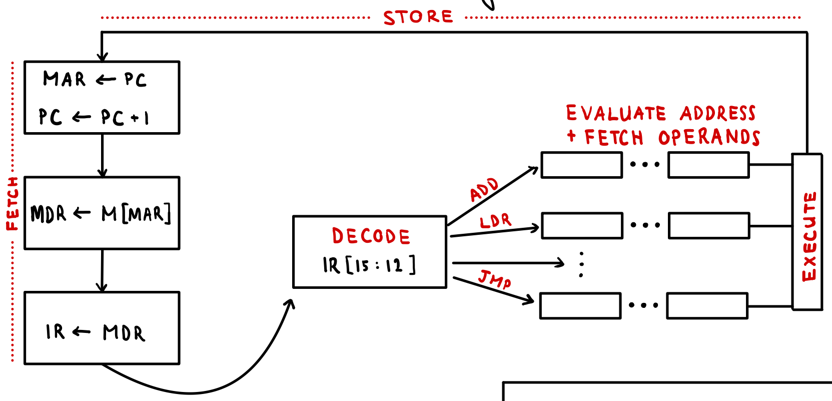
CLOCK GENERATOR CIRCUIT



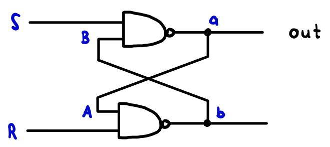
Based on crystal oscillator

Generates Regular Sequence of 0 and 1 logic levels

Control Unit State Diagram



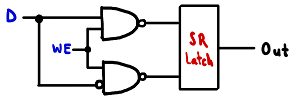
SR - Latch



Used to store 1 bit

- Stands for Set Reset Latch
- Note: Latch is unstable if S=R=0
- Needs Active High Input

Gated D-Latch

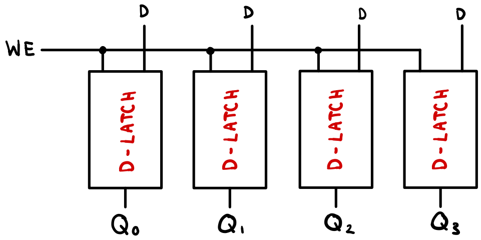


When WE=0, Latch holds data.

When WE=1, latch is set to value of D.

WE: Write enable   D: Data

Registers and Memory Cells



4-BIT MEMORY CELL OR REGISTER

Memory Circuits

→ Bits are stored in array of D Latches

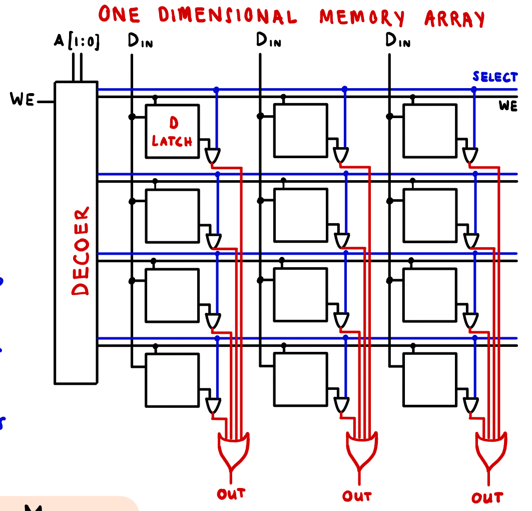
→ An address is decoded to select a row

→ WE specifies if we want to retrieve or store a value

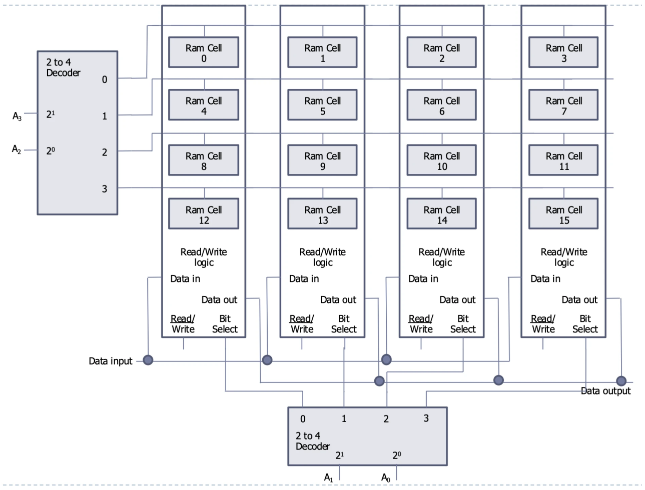
→ To write data, data inputs at top.

→ To read data, data outputs at bottom

ONE DIMENSIONAL MEMORY ARRAY



2 dimensional Memory



Interface to Memory

LOAD from memory

→ Write address of memory into MAR

→ Send a "read" signal to memory

→ Store contents into MDR

STORE to memory

→ Write data into MDR

→ Write the address to write to in MAR

→ Send "write" signal to memory, copy contents from MDR into location at MAR

Common Trap Routines

vector	symbol	method
x20	GETC	reads a character (no echo) into R0
x21	OUT	output a character (R0) to the console
x22	PUTS	write a string to console. R0 contains pointer to string
x23	IN	same as GETC but echos character
x25	HALT	halts the program

System Calls

→ User program invokes system call

→ Operating System code performs operation

→ Returns control to user program

RTI Encoding 1000 000000000000

Operations PC ← copied back from OS stack

TRAP Encoding 1111 0000 trapvector8

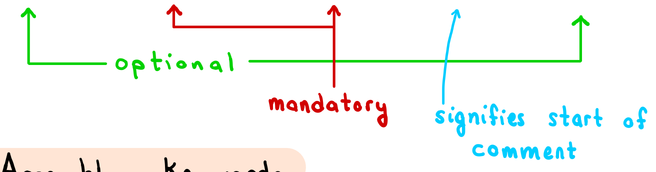
Operations 1) PC copied to OS stack 2) New PC = MEM[trapvector 8]

Subroutines

→ Lives in user space
→ Performs well defined task
→ Invoked by another program
→ Returns control to main program

Assembly Syntax

LABEL OPCODE OPERANDS ; COMMENT



Assembly Keywords

.ORIG	address	specify program start address
.END		
.BLKW	n	allocate n addresses
.FILL	n	fill address with n
.STRINGZ	"string"	allocate n+1 addresses inc null

```
# STACKS
R1 <- Pointer to Stack Pointer
R7 <- Value to Push / Value from Pop

We Push and Pop to the stack using ...
JSR       Pop
JSR       PUSH

**PUSH AND POP**
PUSH      MEM[R1] <- R6
                  R6 <- R6 - 1

POP       R6 <- R6 + 1
          R1 <- MEM[R6]

EMPTY     Specifies intial stack pointer
```

Memory Mapped Controller Registers

Location	I/O Register	
xFE00	Keyboard Status Register	KBSR
xFE02	Keyboard Data Register	KBDR
...	...	

Addressed from xFE00 → xFFFF (is a priveledged zone)

Statuses

Controller

0 means controller can change value and toggle status

1 means controller can't change value

CPU

0 means CPU cannot read value

1 means CPU can read value and change status to 0

JSR

Encoding 0100 1 PCoffset11

Operations 1) R7 ← PC 2) PC ← PC + PCoffset

JSRR

Encoding 0100 0 00 Base 000000

Operations 1) R7 ← PC 2) PC ← Base R

RET

Encoding 1100 000 111 000000

Operations PC ← R7

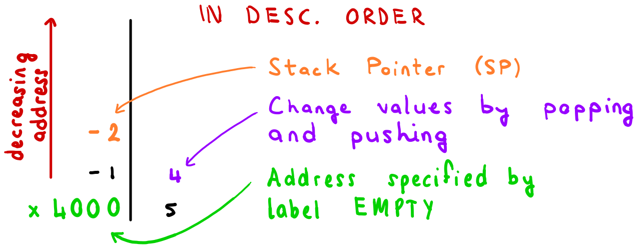
De Morgan's Law

OR FORM $\overline{AB} = \overline{A} + \overline{B}$

AND FORM $\overline{A + B} = \overline{A}\,\overline{B}$

Visualising Stacks

THIS COURSE USES EMPTY STACK CONVENTION IN DESC. ORDER



decreasing address
-2
-1
x4000
4
5
Stack Pointer (SP)
Change values by popping and pushing
Address specified by label EMPTY

note: popping from stack doesn't delete value, it just changes stack pointer

Overflow and Underflow

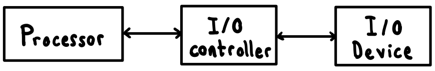
Stack Range: x4000 → x3FFC

Underflow SP > x4000

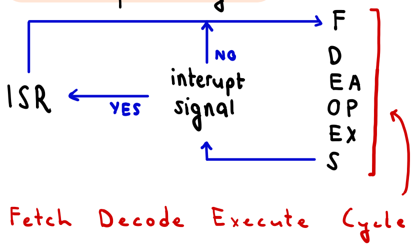
Overflow SP < x3FFC

I/O Controller

→ Provides necessary interface for I/O devices
→ Takes care of low level, device dependant details
→ It buffer data sent to processor so devices remain in sync



Interrupt Cycle



ISR Interrupt Service Routine

Polling

```
# HOW TO USE CONTROLLER REGISTERS IN CODE
POLL      LDI  R1, KBSRPtr
          BRzp POLL
          LDI  R0, KBDRPtr
          ...
KBSRPtr   .FILL xFE00
KBDRPtr   .FILL xFE02

**To Note:**
- Polling is a waste of cycles
```

Interupt Service Routine

When an external device needs handing ...

1) The current program is stopped by the OS and the state is saved.

2) The Interrupt Service Routine is run, satisfying the I/O devices needs.

3) The programs state is restored and gains control over CPU.

- Interupts are assigned a priority, P0 → P6, with 6 being the highest
- Check page before for Interrupt Cycle

Finite State Machine (control unit)

On each machine cycle FSM changes control signals for next phase of instruction processing. Like ...

→ What component drives the bus

→ Which registers are write enabled

→ Which operation should the ALU perform

Global Bus

Set of wires that allow various components to transfer 16 bit data to other components.

One or more components may read data from the bus on any cycle

LC3 Data Path should be on exam appendix