

Novasaur TTL Retrocomputer

Building a Computer
From Scratch

VCF East 2023



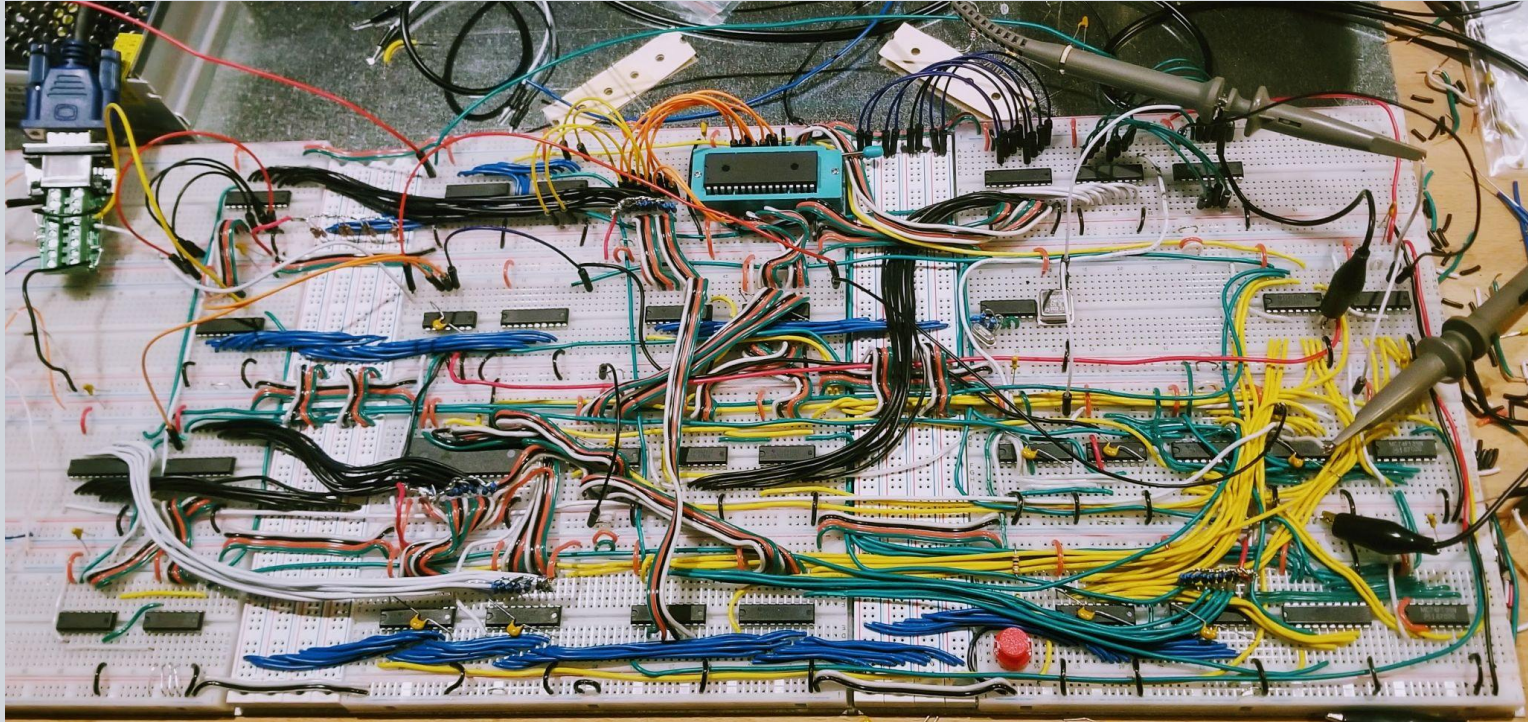
**SOLID-STATE
MACHINES**

40 Years in the Making

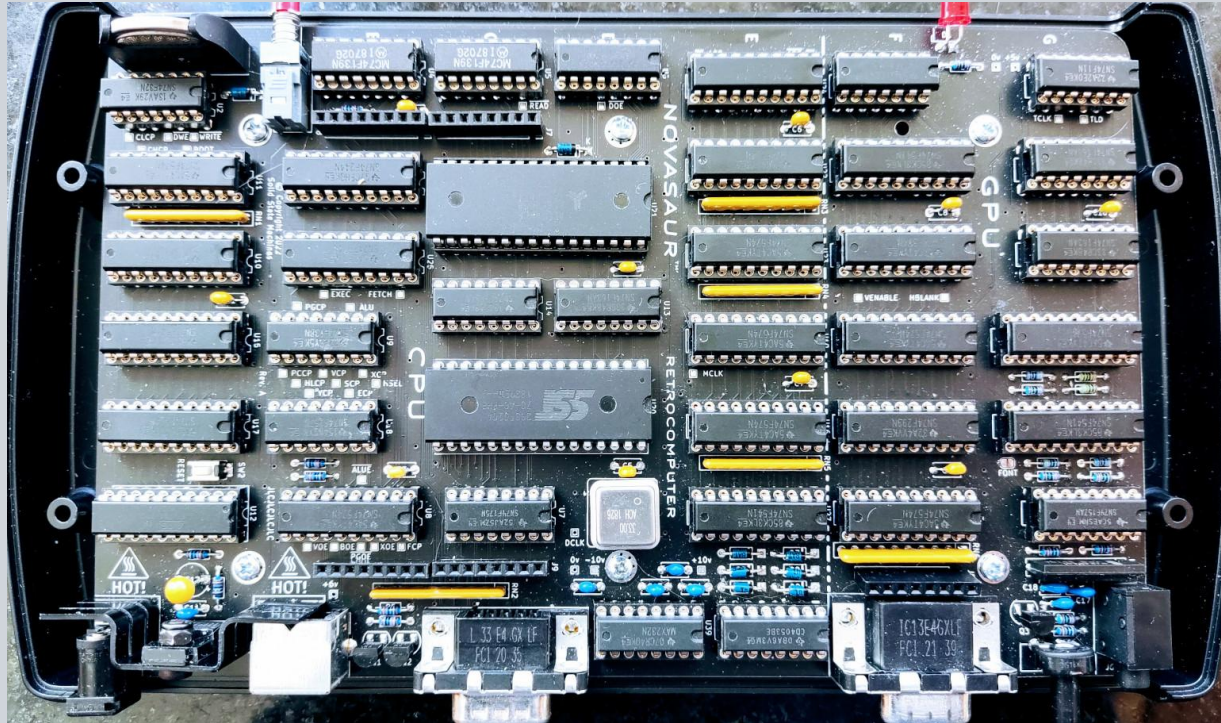
- **1979** - First logic circuit (half adder)
- **1983/4** - Original CPU design
- **1986** - Digital Systems Engineering
- **1991** - Production Engineering
- **2018** - Gigatron release
- **2019** - Novasaur begins



Novasaur Rev '0'



Novasaur Rev 'A'



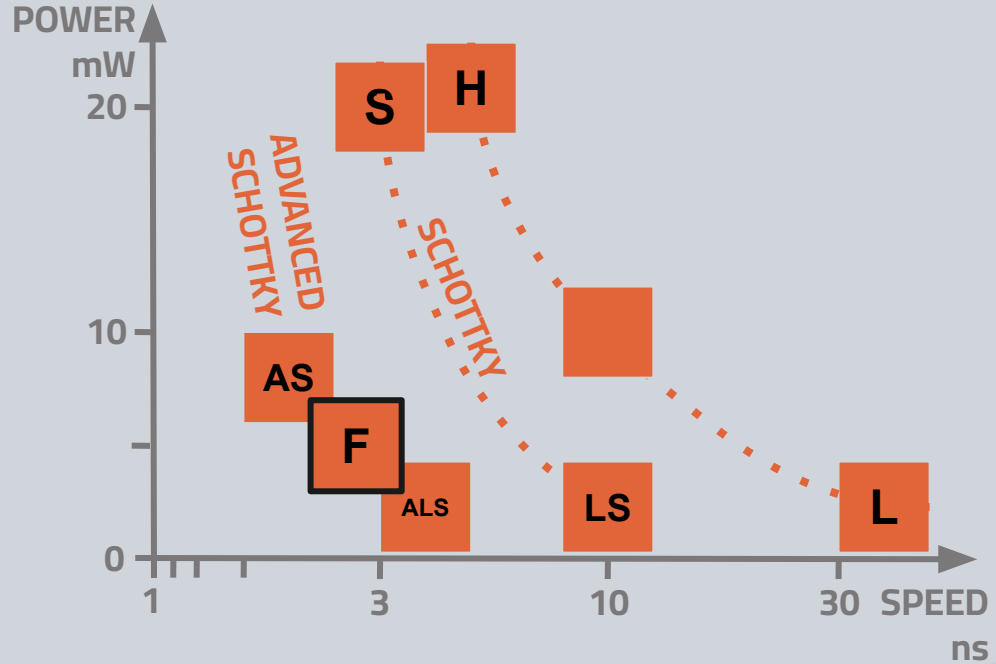
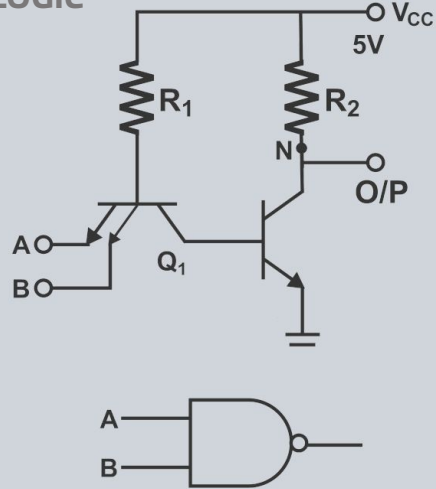
Hardware Requirements

- Simple building blocks
- 1970's technology - exists EOY 1979
- Readily available - 50+ years later?
- 74-series TTL chips - not CMOS
- *Scorched-earth* minimalism
- Fast enough to be useful



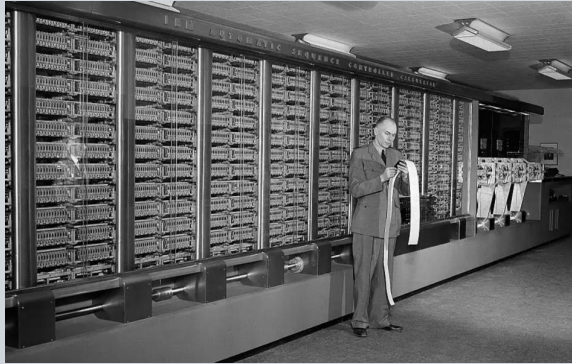
Technology

TRANSISTOR-TRANSISTOR
LOGIC



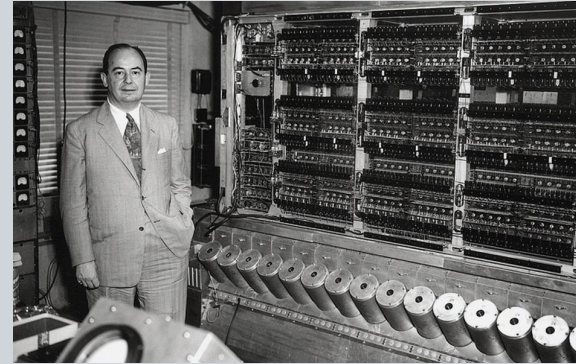
Computer Architectures

Harvard



- **Harvard Mark 1**
- Punch tape is the program memory

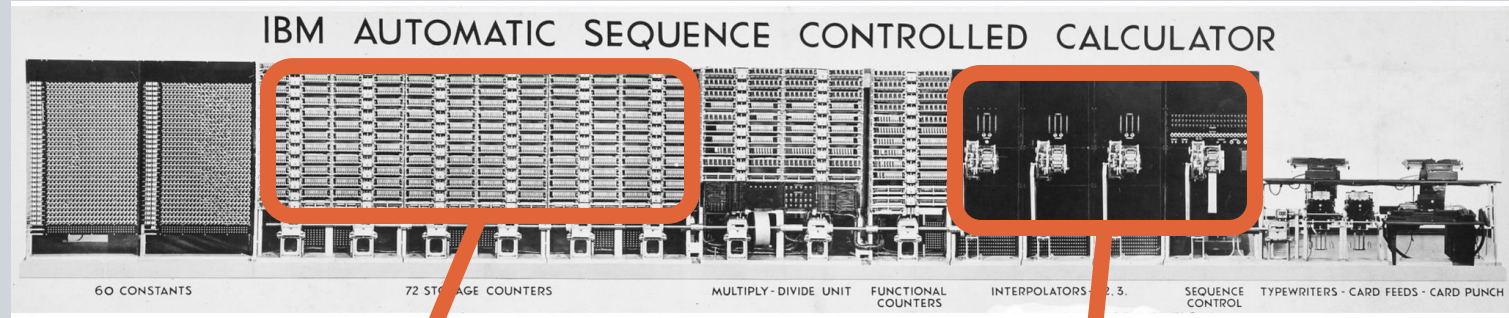
von Neumann



- **EDVAC**
- Stored program (loaded from tape)



Harvard Mark 1



Data Memory

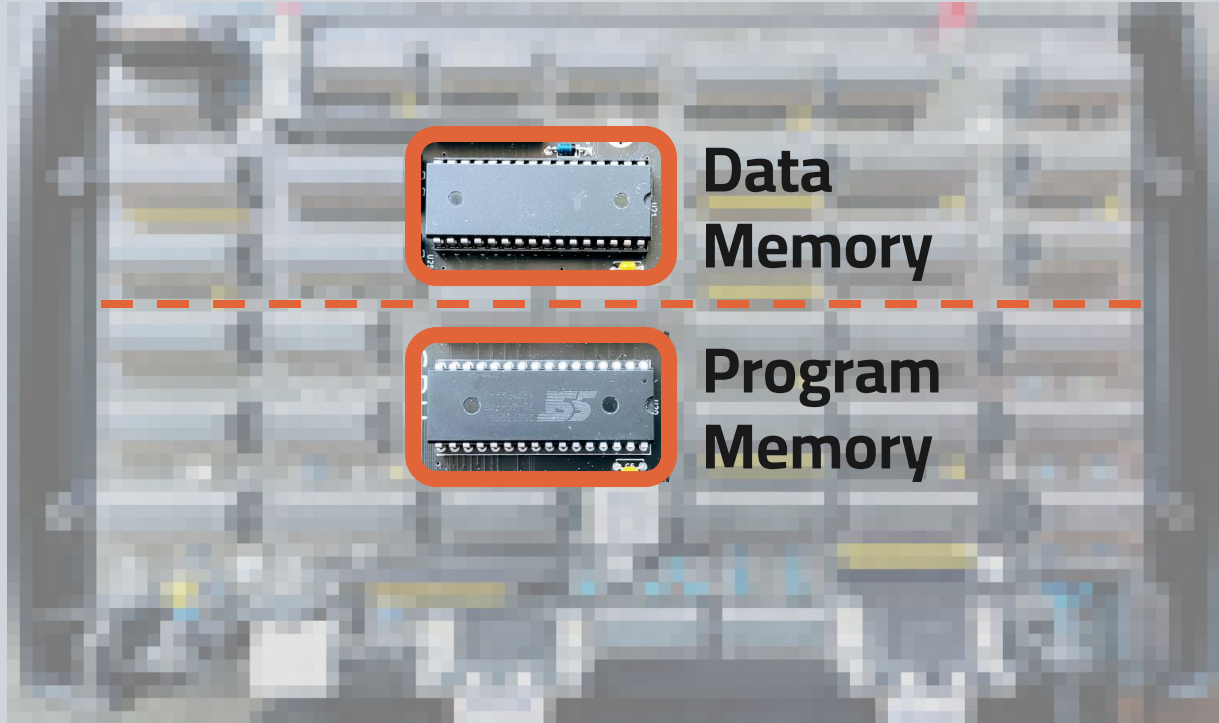
Storage Counters
(23 decimal digits)

Program Memory

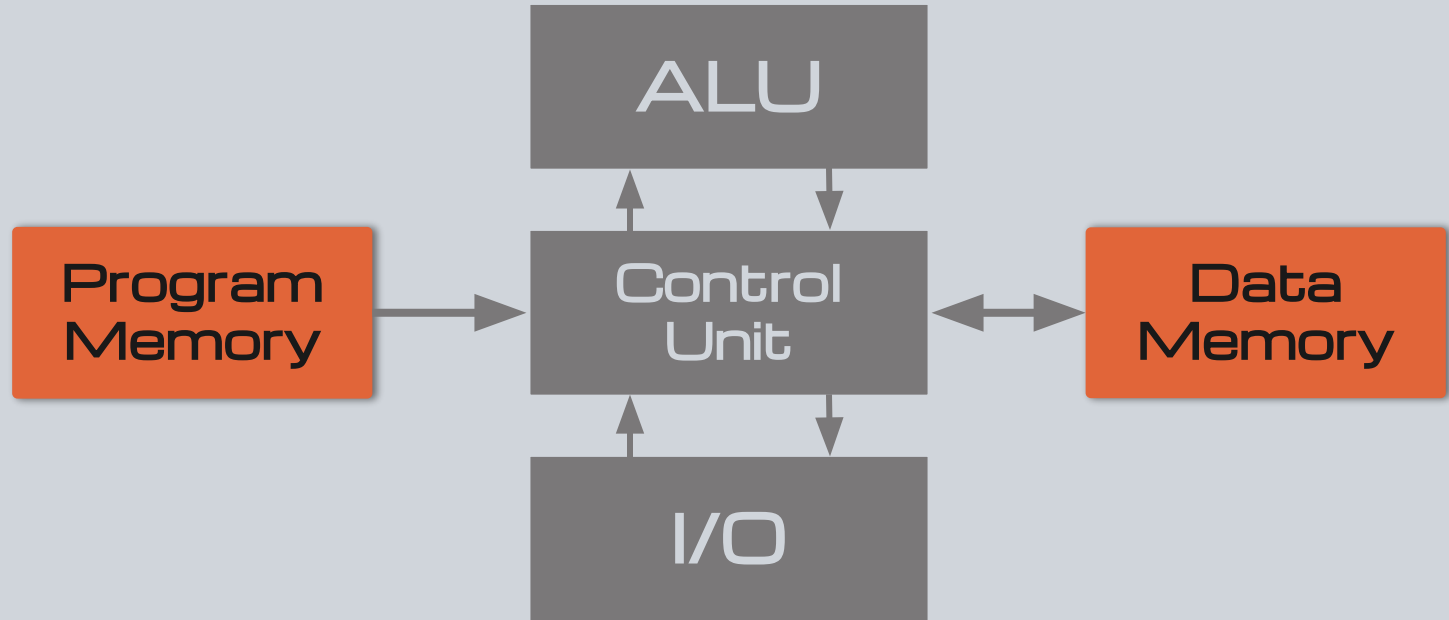
Paper Tape
(24 bit instructions)



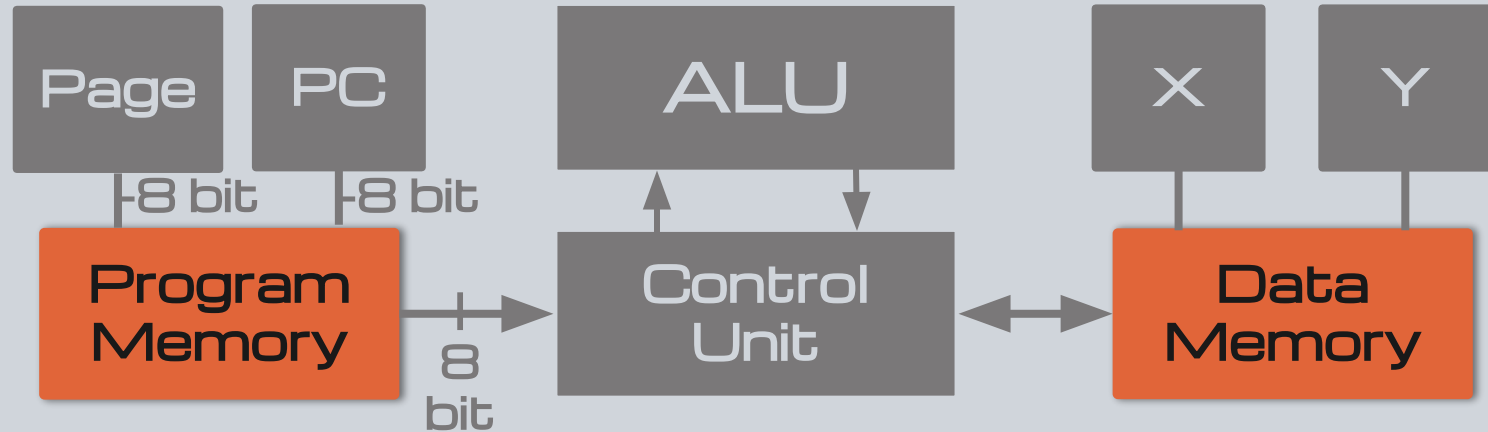
Harvard Architecture



Harvard Architecture



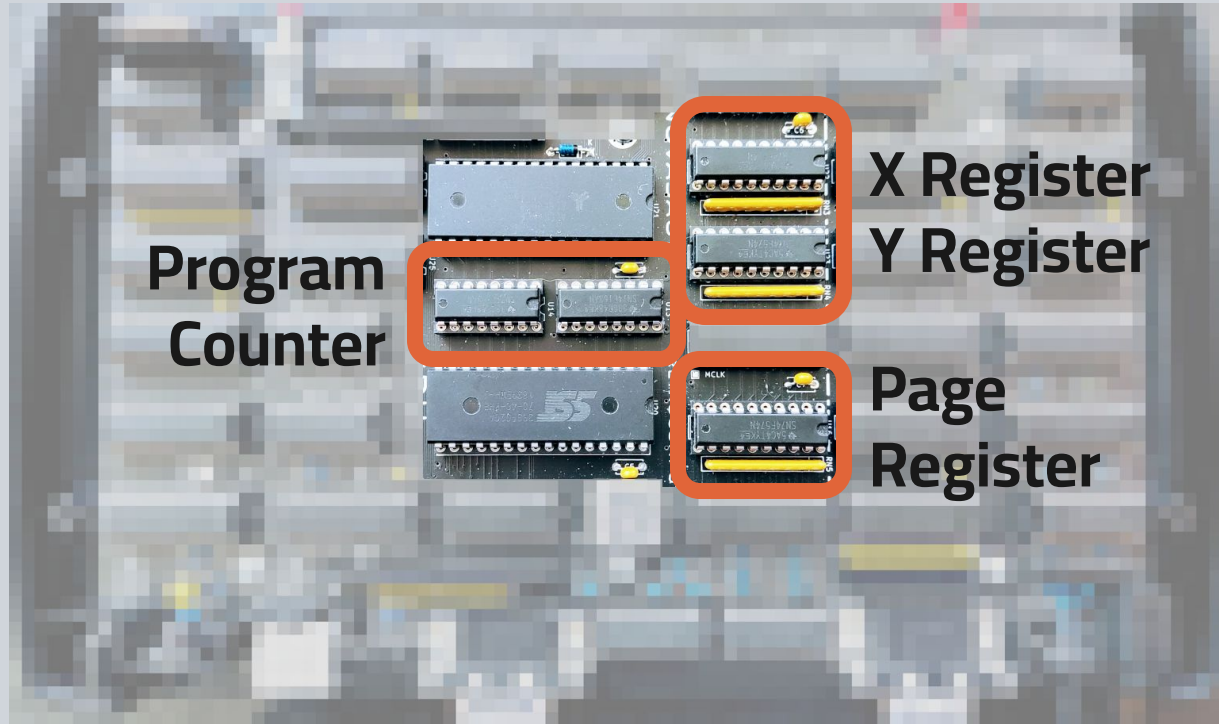
Simple Addressing



- 8-bit **page** register
- Single 8-bit counter
- No address bus needed
- *one index per memory
- 8-bit X register
- 8-bit Y register



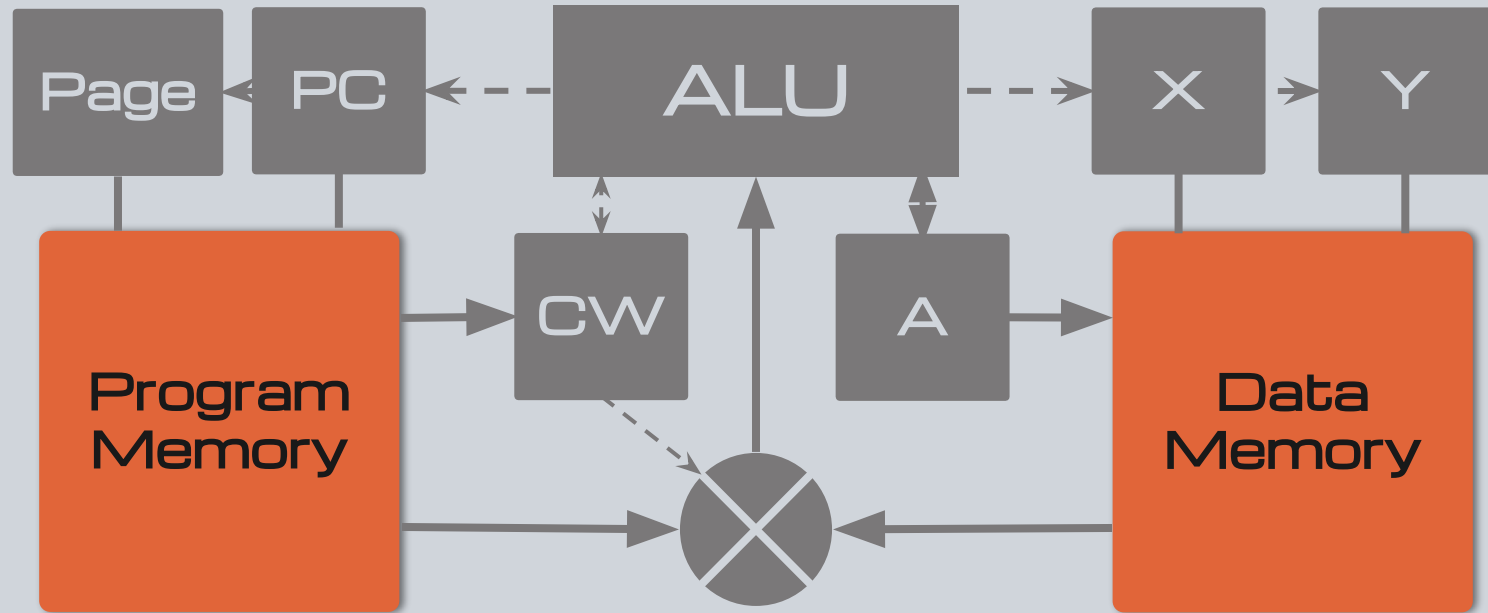
CPU Addressing



Minimal 'Toy' Computer

- Instructions are represented as a **Control Word**
- The Control Word defines:
 - A data **Source** - program or data memory
 - An **ALU** operation to perform - add A or 0
 - A **Destination** - any register and/or data memory
- Every instruction is an ALU operation
- Every register takes its input from the ALU

Does it Work?



Can it Increment the X Register?

Assume the value is saved in memory address [33, 55]

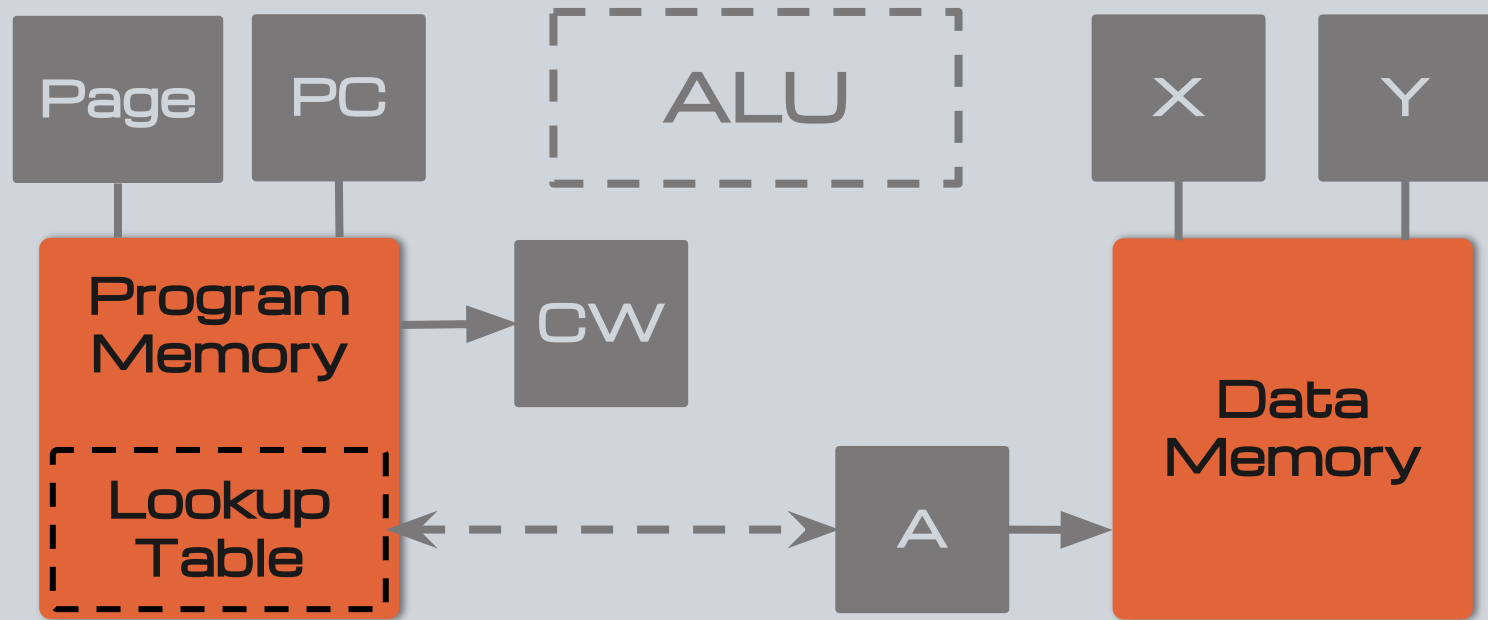
- $X = 55 + 0$ # set X index of stored value
- $Y = 33 + 0$ # set Y index of stored value
- $A = 1 + 0$ # set up an increment function
- $A = [Y, X] + A$ # increment stored value
- $A \rightarrow [Y, X]$ # update stored value
- $X = 0 + A$ # set X with incremented value

Can it be Improved?

*Assume the value is saved in memory address [33, **FF**]*

- ~~$X = 55 + 0$~~ # use a "zero page"
- $Y = 33 + 0$ # set Y index of stored value
- $A = 1 + 0$ # set up an increment function
- $A = [Y, X] + A$
- $A \rightarrow [Y, X]$ # combine as $[Y, FF], X = [Y, FF] + A$
- $X = 0 + A$

Can it get Smaller?



Lookup Table Options

Fn

- 8x8-bit addr
- **Pro** - simple
- **Con** - large
- 64k per function

Fn

- 4x4-bit addr
- **Pro** - small
- **Con** - complex
- 2 cycles, carry, intermediate result register

Fn

- 8x4-bit addr
- **Trade off**
- 2 cycles, no carry, A is intermediate, 8k per function



Example 8x4 Table Addition

A =

2	9
	8

HL =

5	8
---	---

+L

Lookup 1 :

3	1
5	

A = A + L

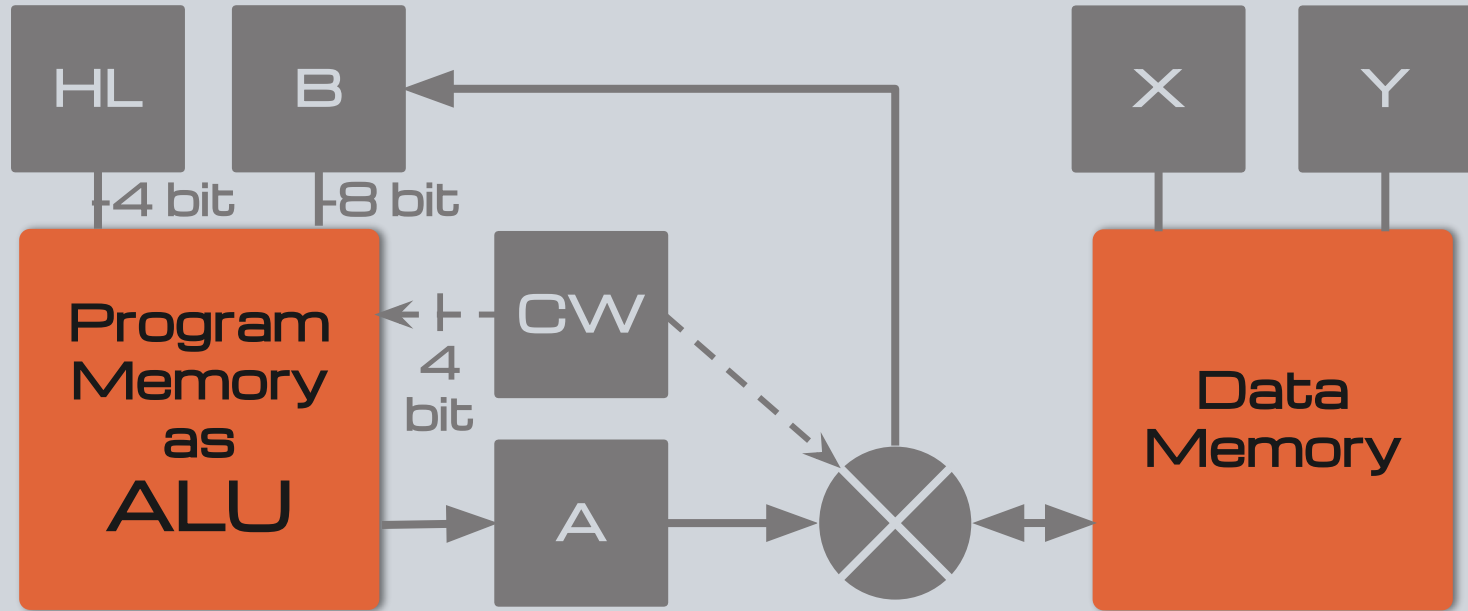
+H<<4

Lookup 2 :

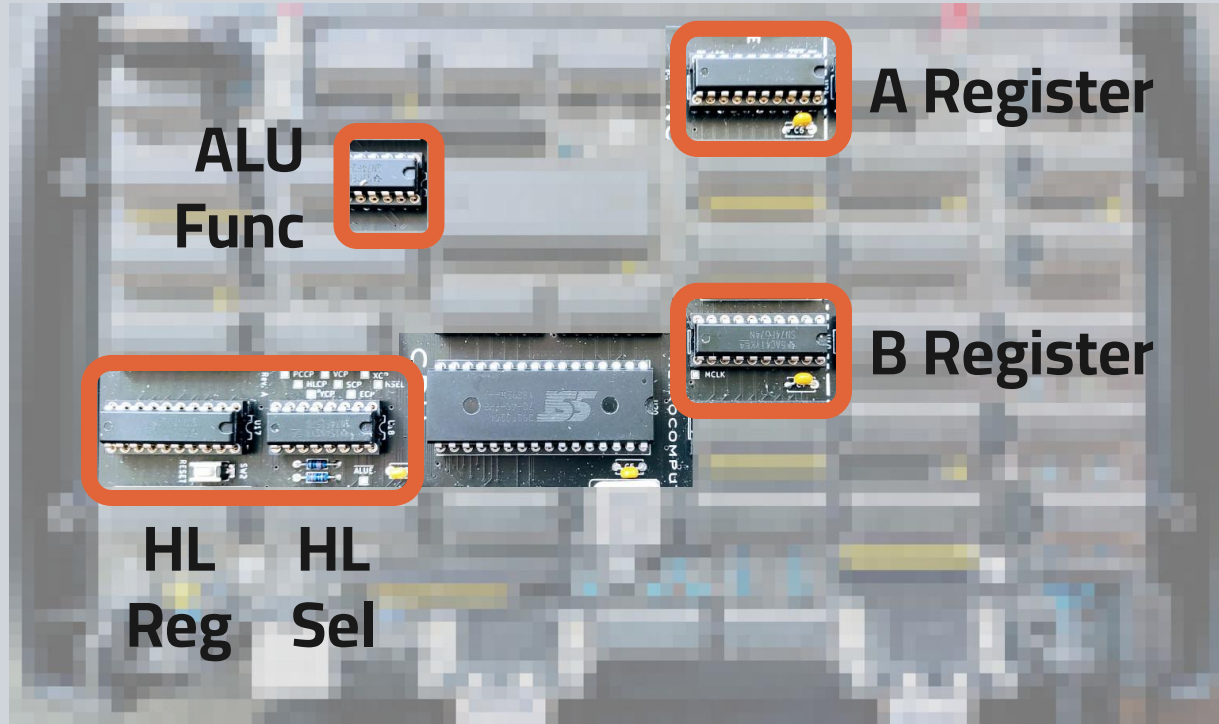
8	1
---	---

A = A + H<<4

ALU Addressing



ALU Addressing



ALU Functions

- **MV** - move HL to dest
- **ADD** - addition (no carry)
- **AF** - arithmetic flags
- **AND** - logical AND
- **OR** - logical OR
- **XOR** - logical XOR
- **WAV** - wave table
- **DIS** - dispatch page
- **MUL** - multiply ($8=4*4$ bits)
- **DIV** - divide ($4=8/4$ bits)
- **COM** - comms state
- **VID** - video timing
- **FNC** - unary functions
- **FND** - unary (8080)
- **FNE** - unary (VM)
- **FNF** - unary (default)

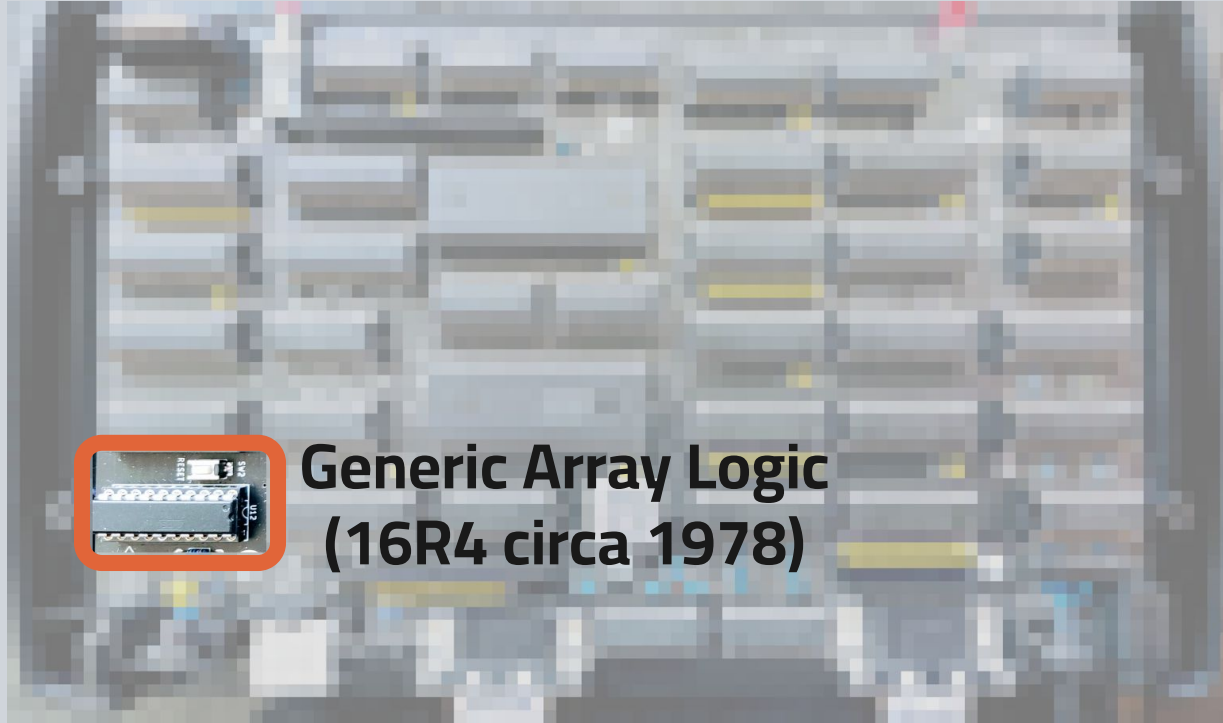


Default Unary Functions

- **IDEN** $A = A$
- **INC** $A = A + 1$
- **INC2** $A = A + 2$
- **DEC** $A = A - 1$
- **1COM** $A = !A$
- **2COM** $A = -A$
- **LSR** $A = A \gg 1$
- **LSL** $A = A \ll 1$
- **ZERO?** $A == 0?$
- **KERN?** $\text{bank} == 1?$
- **INCLINE** inc VM line
- **INCBLK** inc VM block
- **NSR** $A \gg 1 \mid 0x80$
- **SWAP** $AB = BA$
- **REVERSE** reverse bits
- **AUDIO** flip sign/reverse



Execution State Machine



**Generic Array Logic
(16R4 circa 1978)**

Execution Sequence

FETCH

Read
control
word from
**Program
Memory**

READ

Decode
and
Read data
from
**Data
Memory**

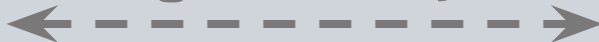
EXECUTE

Read ALU
or
immediate
value from
**Program
Memory**

WRITE

Write result
to register
and/or
**Data
Memory**

Single CPU cycle



Memory Concurrency

FETCH

ROM

READ

RAM

READ

RAM

LOOKUP

ROM

EXECUTE

ROM

READ

RAM

WRITE

RAM

N/A

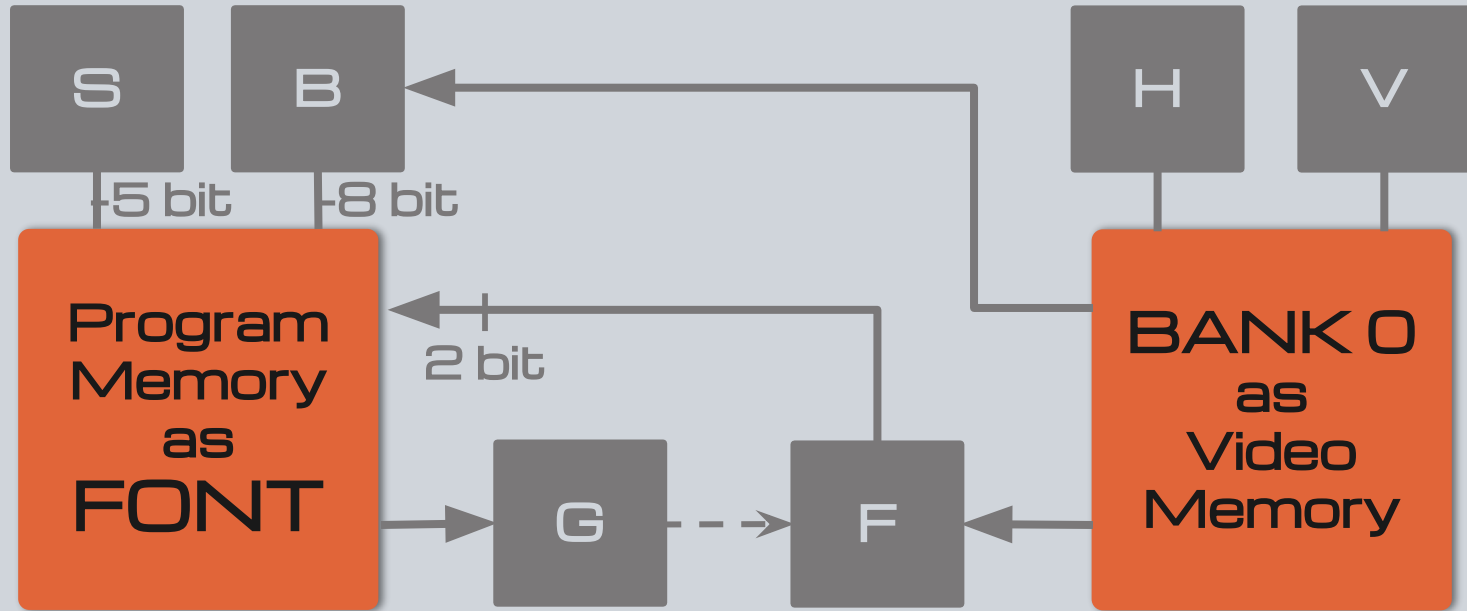
ROM



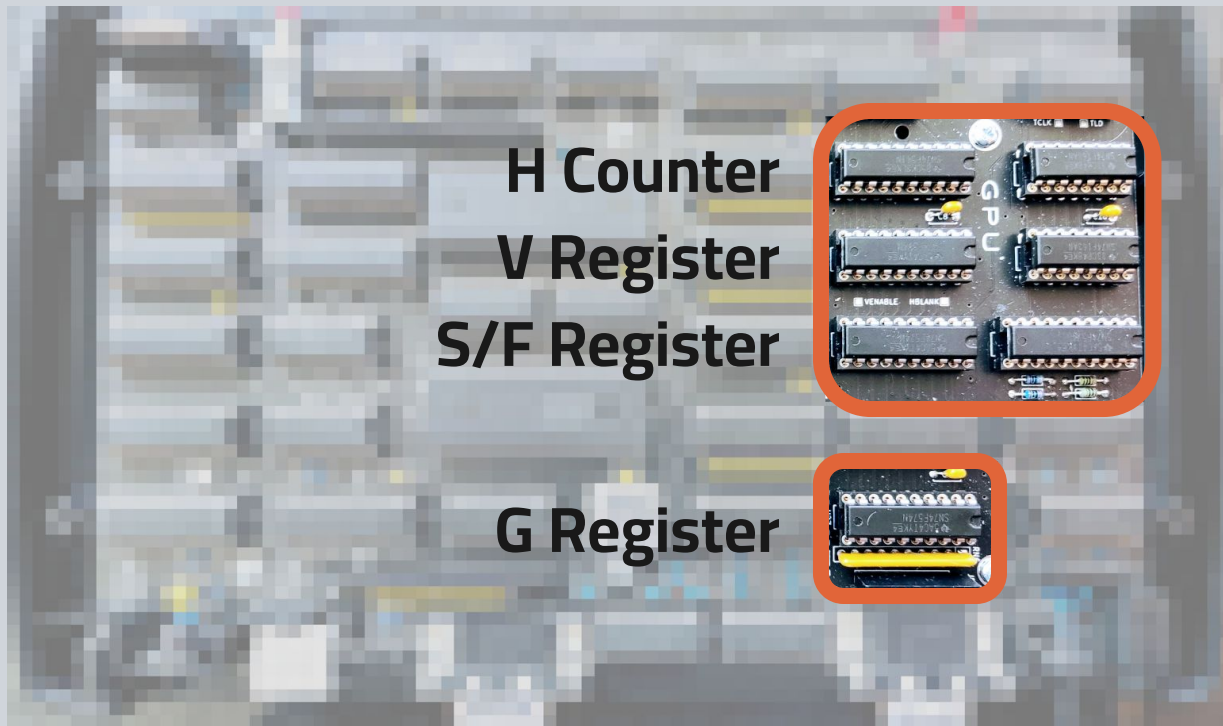
Introducing the " GPU "

- **Transparent DMA** - read RAM while CPU reads ROM
- Add **Horizontal** counter and **Vertical** index register
- **Graphics Mode** - one byte per pixel (256 colors)
- **Text Mode** - two bytes per char (ASCII, 8 colors)
 - Read character code point from **RAM**
 - Lookup glyph data in **ROM** → shift register
 - Read FG/BG/Font value from **RAM**

GPU Addressing



GPU Addressing



256k ROM Memory Map

BANK 0

Program
64k

BANK 1

Storage
64k

BANK 2

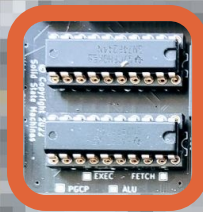
ALU
high

Font

BANK 3

ALU
low

Input/Output



Ei Buffer
Eo Register

Line : Analog
Driver : Switch



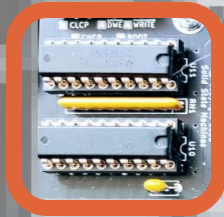
Ei/Eo Register



512k RAM Memory Map

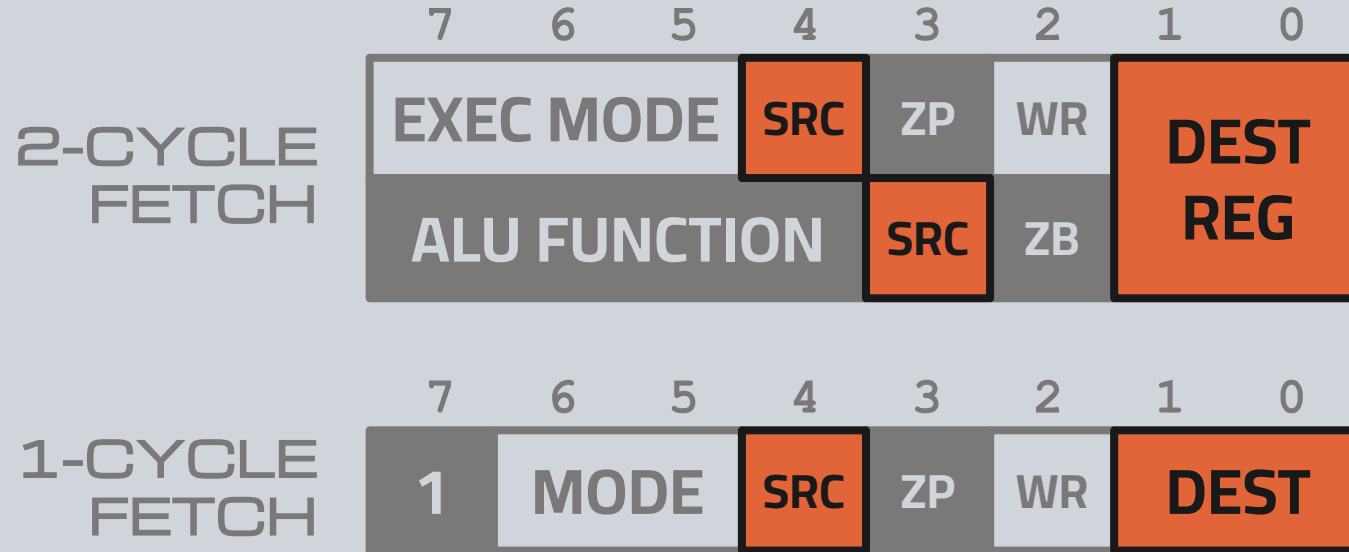


Control Word



**Control
Word**

Control Word



Execution Modes

0	4-cycles	ALU	HL→A
1	3-cycles	ALU	L
2	3-cycles	ALU	H → A
3	3-cycles	ROM	H
4,5*	1 or 2-cycles	LOAD	* conditional on A sign
6,7	2-cycles	FNF	H → A, H

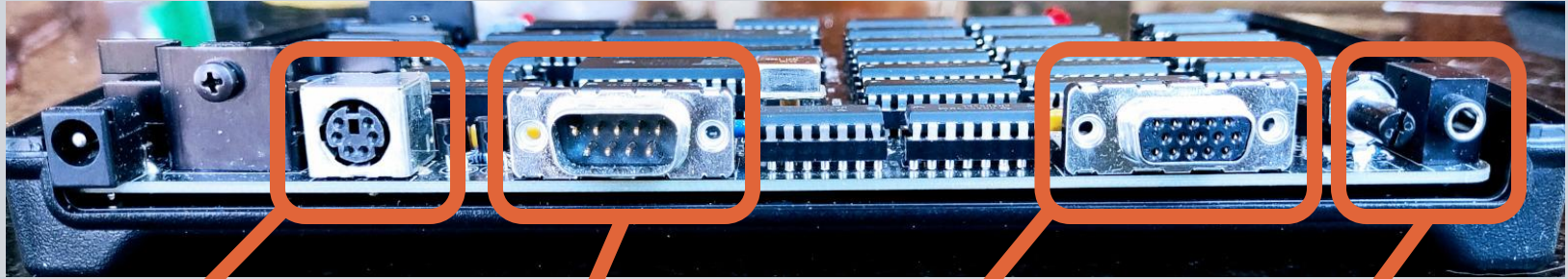


Increment the X Register

Code	Assembler	Description	Cycles
9433	LD Y, 0x33	Immediate load Y	2
9510	LD HL, 0x10	Immediate load HL	2
58FE	FNFH DZ, XD	[Y,FF] , X = FNF (H , [Y, FF])	3
6 Bytes		~850ns @ 121ns/cycle	7 Cycles



How does it actually work?



Keyboard

PS/2
US/English

Serial

RS232
CTS/RTS

Video

VGA, SVGA, XGA
60,75Hz Refresh

Audio

3 Voice, 1 Noise
ADSR



Software Requirements

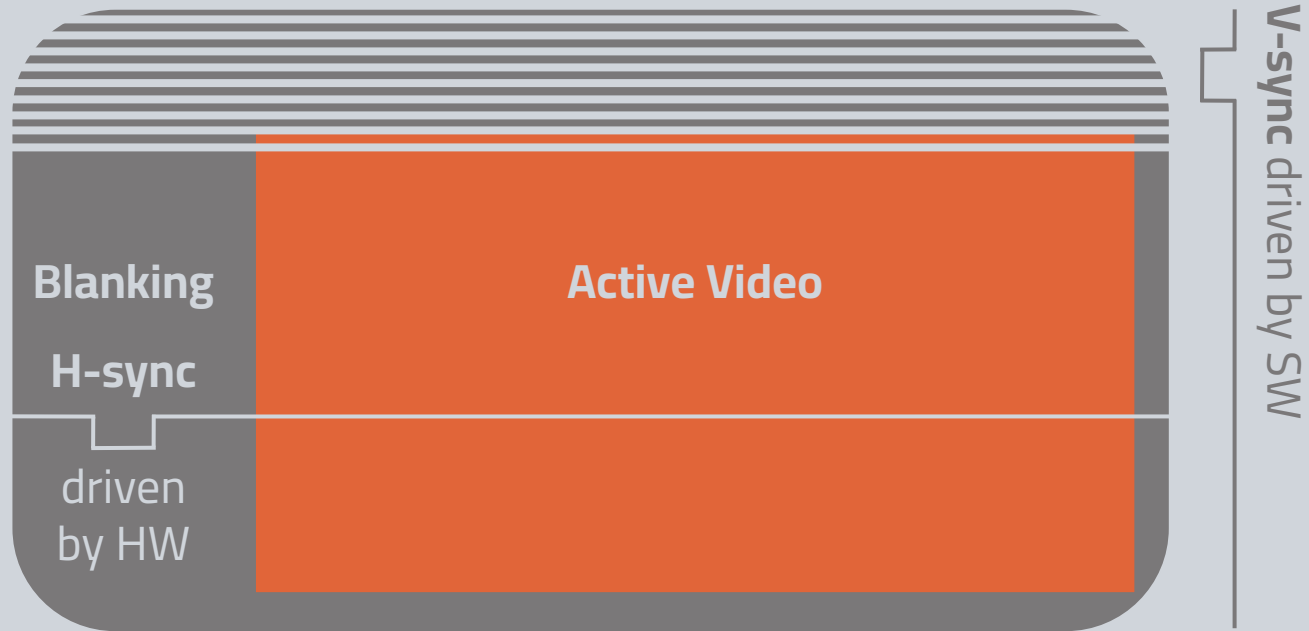
- **Concurrent protocols** - lots of state
- **Critical timing** - lots of control
- **No Interrupts** - protocols too fast to restart
- **Cannot Yield** - no user program on hardware
- **Stored Program** - user program → data memory
- **Interpreter** - virtual CPU with existing ISA



Software Stack



Video



Video Modes

Mode	Resolution	Text Columns	Lines
VGA	640 × 480	104	525
VGA @ 75	640 × 480	80	500
SVGA	800 × 600	80	628
XGA	1024 × 768	64	806



Video Control

- **Update indexes** - $S + V$ - reset H
 - **Count lines** - 500-806 - reset V
 - **Line count** > 9-bits
 - **Use ALU lookup**
 - **Block** count - 8 bits
 - **Line** count - 4 bits
- } **VID** function



Blocks of Lines

BLOCK 0	VID	COM	Reset Count	0	LINE
	VID	COM		1	
	VID	COM		2	
	VID	COM		3	
BLOCK 1	VID	COM	Count	0	LINE
	VID	COM		1	
	VID	COM		2	
	VID	COM		3	



Virtual Machine Cycle

- **Break line into columns**
 - **VGA** (31.5 kHz) ~ 192 kHz / 6
 - **SVGA** (37.9 kHz) ~ 192 kHz / 5
 - **XGA** (48.4 kHz) ~ 192 kHz / 4
- **Speed limit** - 55ns memory = 18 MHz
- **Use 16.5 MHz** - process cycle = 8.25 MHz
- **43-cycle VMC** - 191.86 kHz



VGA

S	B	1	2	3	4
S	5	6	7	8	9
S	10	11	12	13	14

18 VMCs/block, **10659** blocks/second, **175** blocks/frame



SVGA

S	B	1	2	3
S	4	5	6	7
S	8	9	10	11
S	12	13	14	15

20 VMCs/block, **9593** blocks/second, **160** blocks/frame



XGA

S	B	1	2
S	3	4	5
S	6	7	8
S	9	10	11
S	12	13	14

20 VMCs/block **9593**
blocks/second
161 blocks/frame



Virtual Machine State

VMS
BYTE

MODE LINE

EXT

CYCLE

Line count

Column Count

- 0,1,2
- 3,4,5,6
- 7,8,9,10
- 11,12,13,14,15

0,1,2,3,4,5
0,1,2,3,4
0,1,2,3,4
0,1,2,3



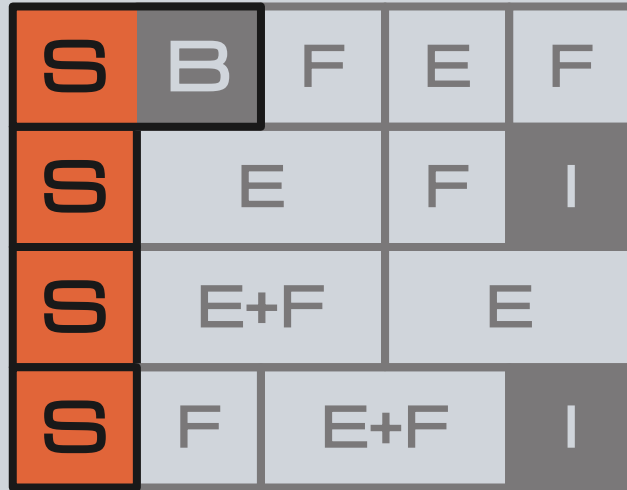
VM Dispatch



- Each page is VM operation
- 84 pages for 8080
- **Page = DIS (VMS, INST)**



vCPU Execution Sequence



1 VMC **FETCH** + 1,2,3 VMC **EXECUTE** (~2.3 VMCs per 8080 instruction)



Threads

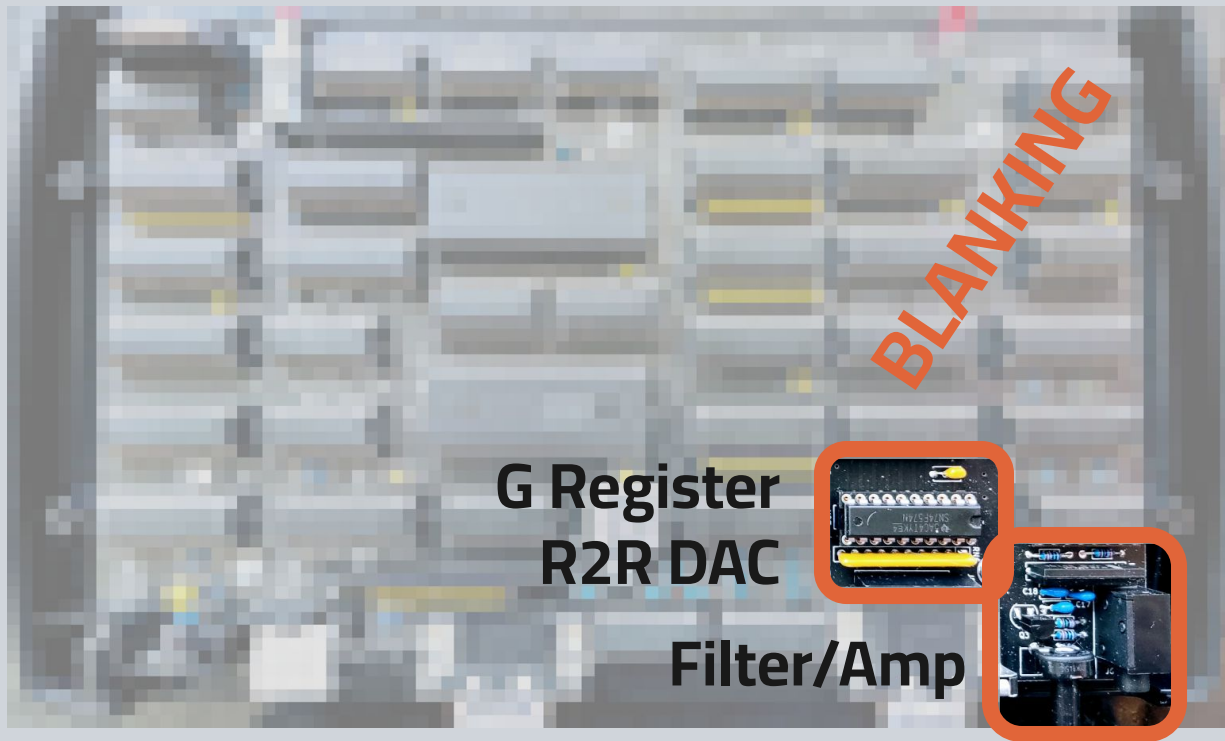
reset **Frame** (60/75 Hz), start **Keyboard** scan (15 Hz), update **Clock**



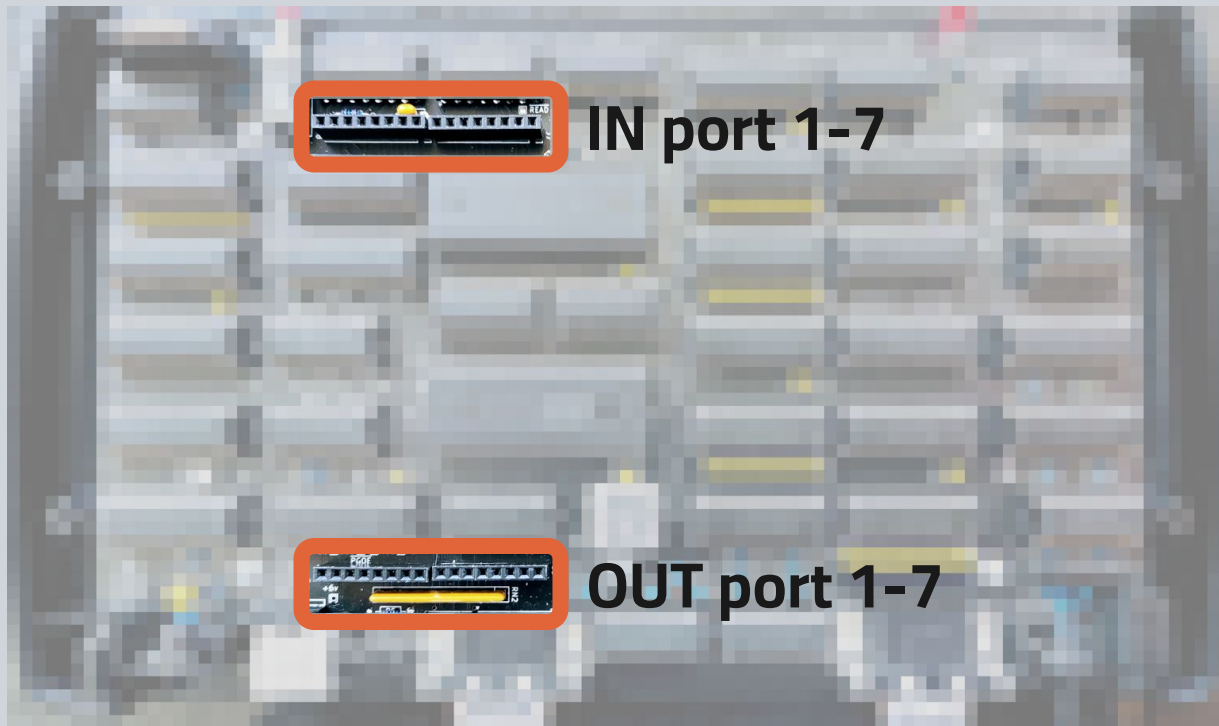
- 6 secs
- 12 mins
- 24 hrs



Audio



Expansion

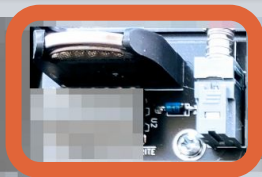


Hardware Abstraction

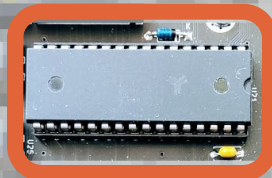
Port#	Input	Output	Notes
8	Serial Rx (TTY)	Serial Rx (TTY)	
9	Console (KBD)	Console (CRT)	Text mode
10	KBD Scan Code	Set Audio Mode	Resource control
11	Character @ cursor pos	Enable/Disable Serial	Resource control
12-55	Zero Page Read	Zero Page Write	Audio, console params
56-63	Zero Page Read	N/A	Read only params (RTC)



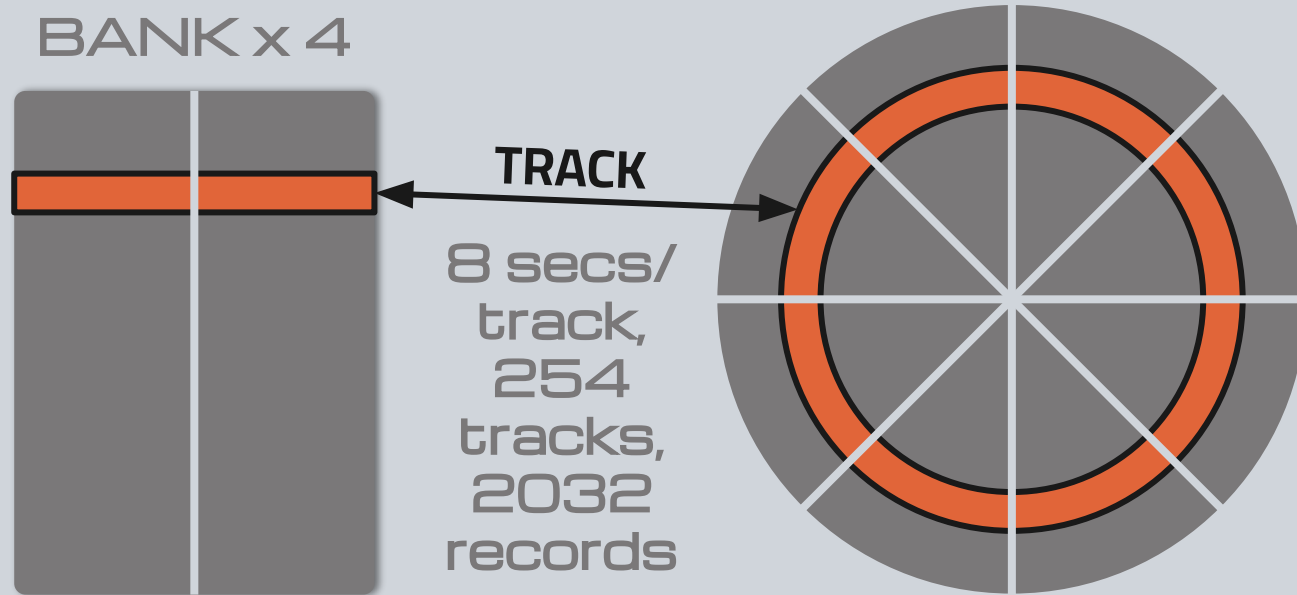
RAM Disk



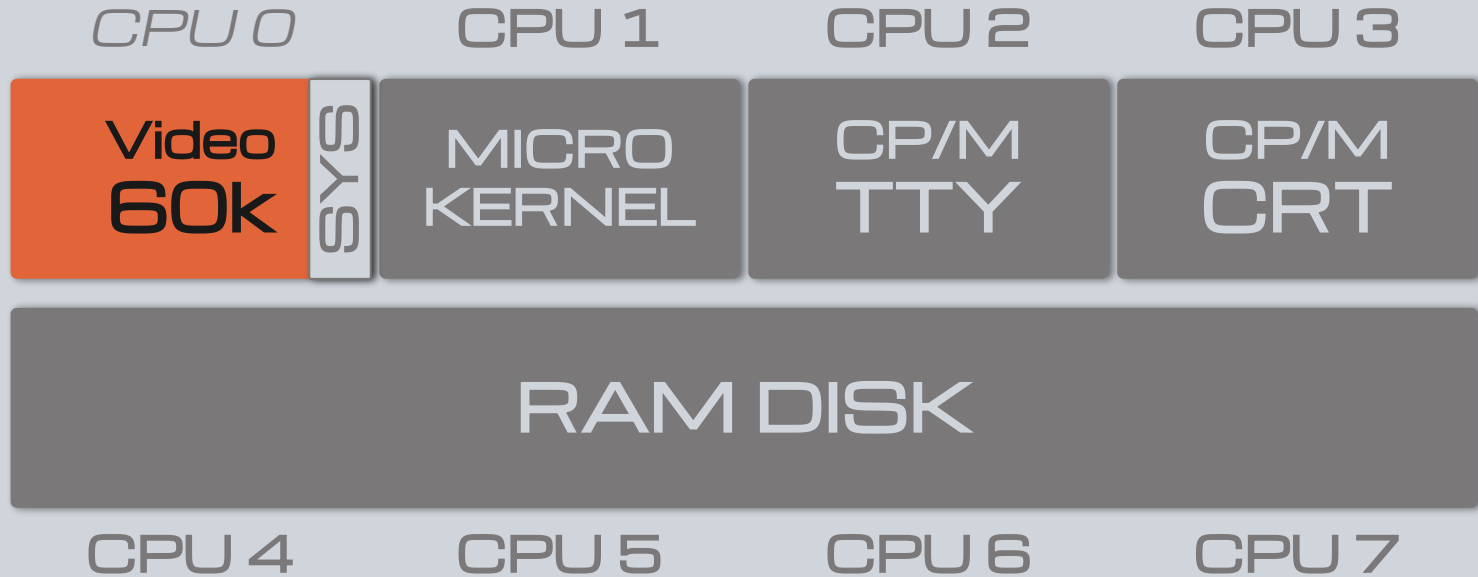
CR2032 @ 4uA



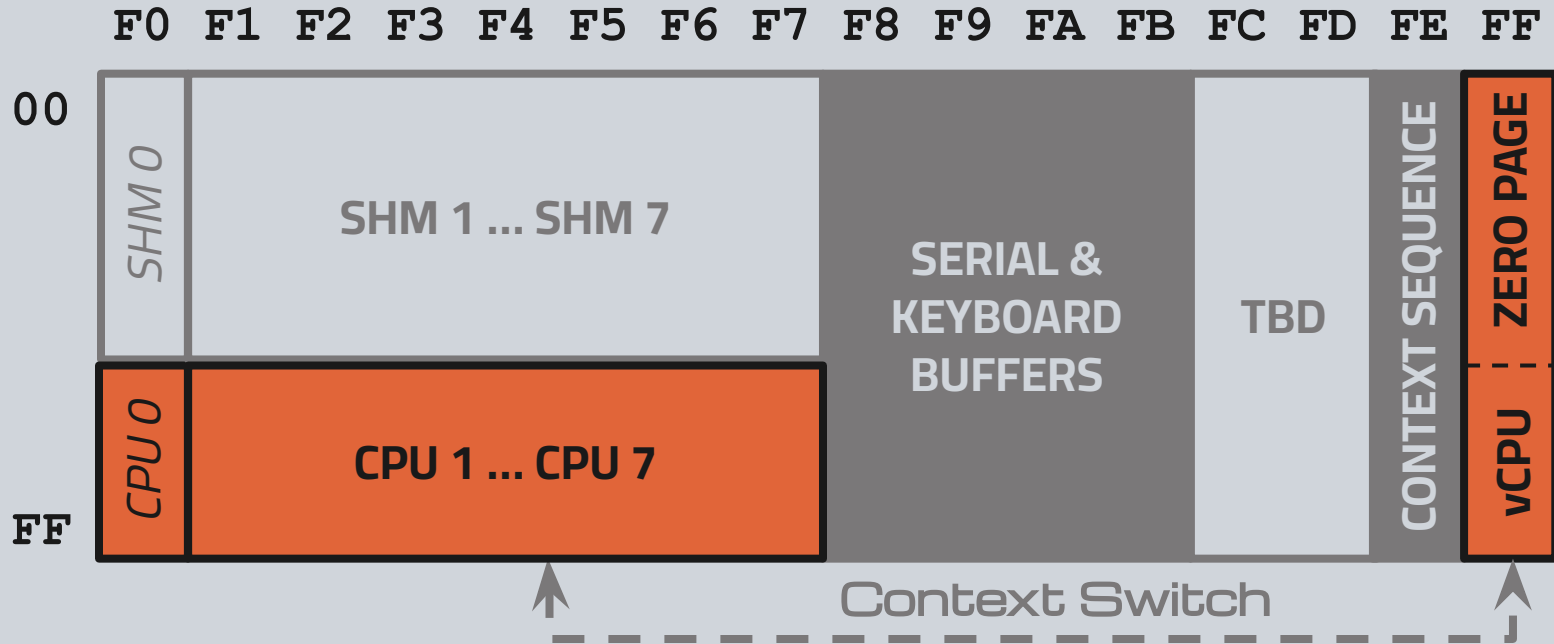
A: Drive RAM Disk



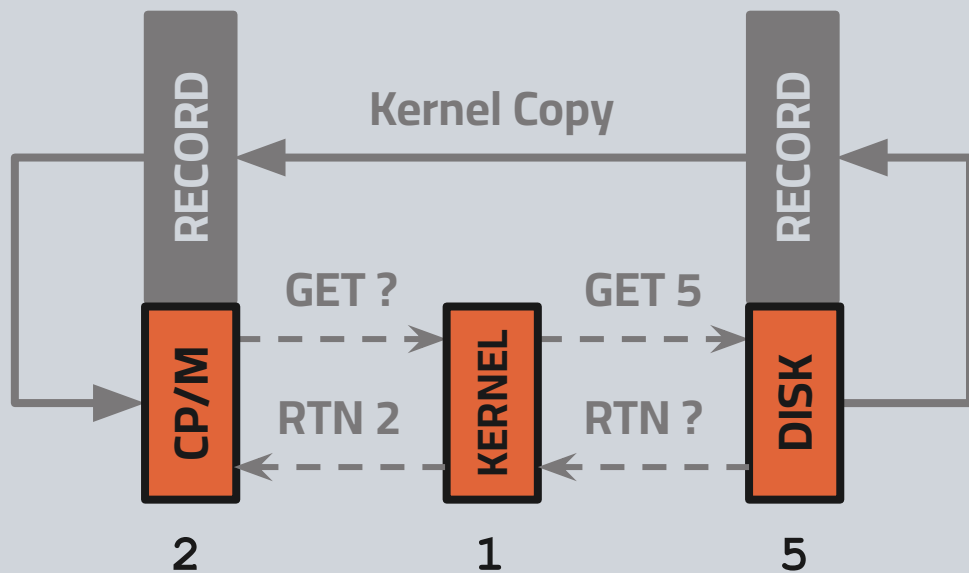
Memory Management



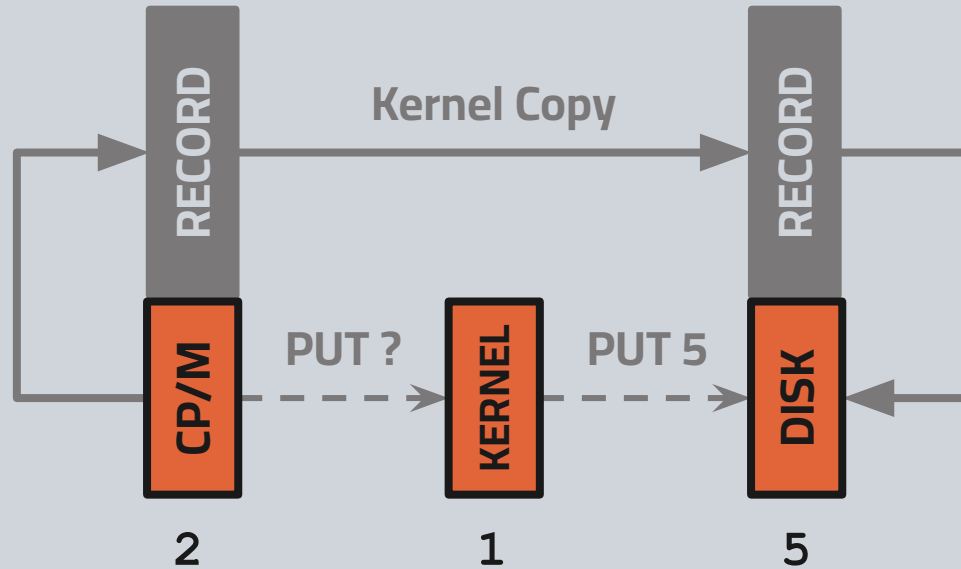
System Memory (Bank 0)



A: Drive Read



A: Drive Write



B: Drive

- **ASM**
- **BASIC** - IMSAI 8k BASIC
- **CPM22** - CCP/BDOS/BIOS
- **DDT**
- **DUMP**
- **ED**
- **LOAD**
- **PIP**
- **STAT**
- **SUBMIT**
- **UTIL** - format, file transfer
- **XSUB**
- **Boot** - bootstrap
- **Disk** - disk client
- **Kernel** - 2k Kernel
- **Monitor** - 2k 8080 Monitor



Novasaur Kit



Thank You!

VCF East

<https://www.solidstatemachines.com>

<https://hackaday.io/project/164212-novasaur-cpm-ttl-retrocomputer>

<https://github.com/ajhewitt/novasaur>

VCF East 2023



**SOLID-STATE
MACHINES**