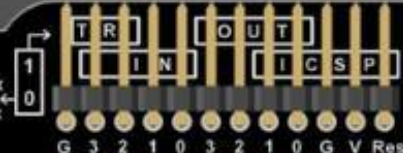


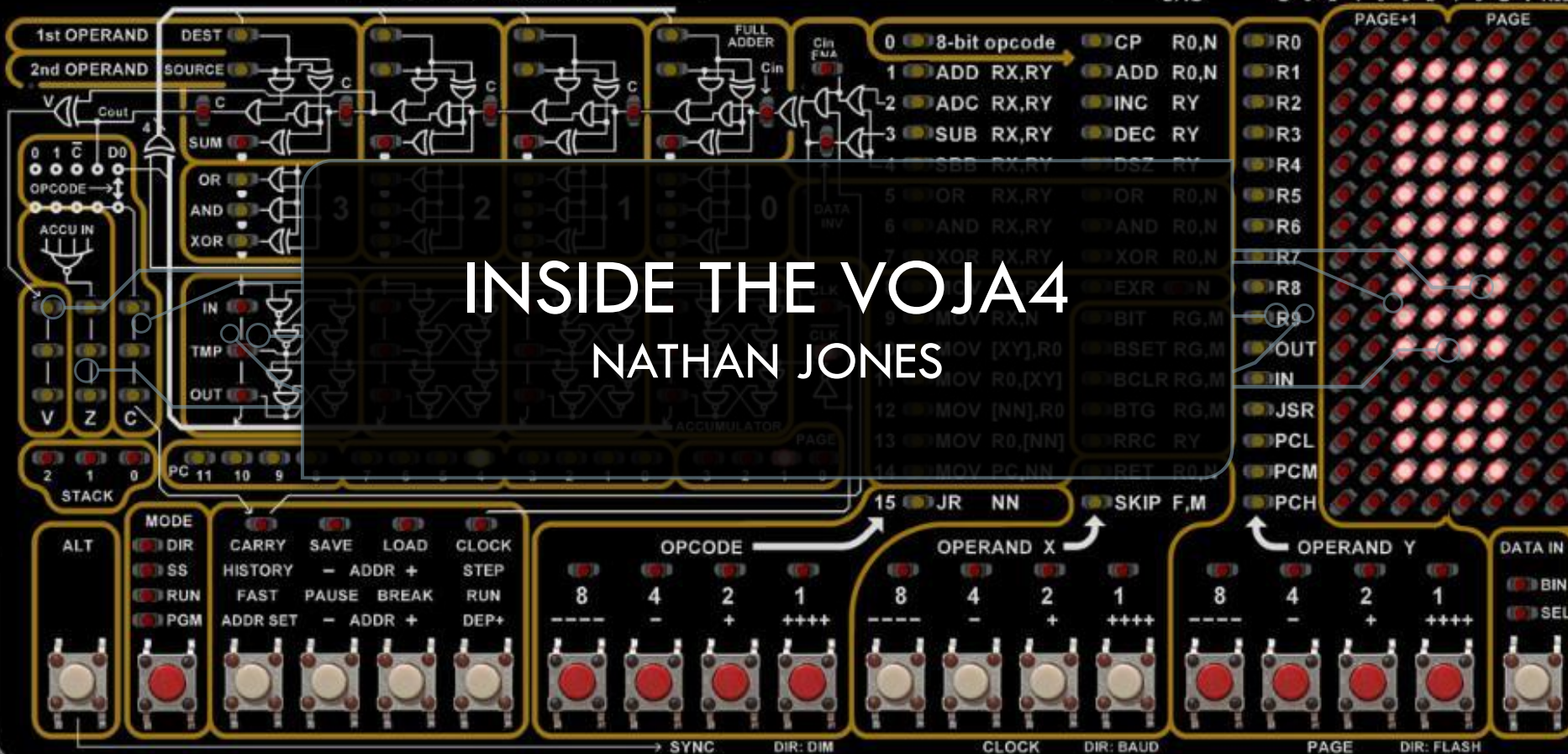
# Supercon.6

November 2022 / Pasadena, CA

3V  
GND  
SAO



## INSIDE THE VOJA4 NATHAN JONES



# TODAY'S GOALS

## Design of the Voja4

### ISA

- Registers
- Memory map
- Instructions

### Trade-offs

## Assembly language programming

### Idioms

- 16-bit adding
- Loops
- Conditionals
- Subroutines

### Pushing the limits

- Demon Attack
- Terti
- Linux on an Arduino
- “Voja4 XL”

# TODAY'S GOALS

Design of the Voja4

Assembly language  
programming

ISA

Trade-offs

Idioms

Pushing the  
limits

Why?

Security, correctness,  
performance, etc

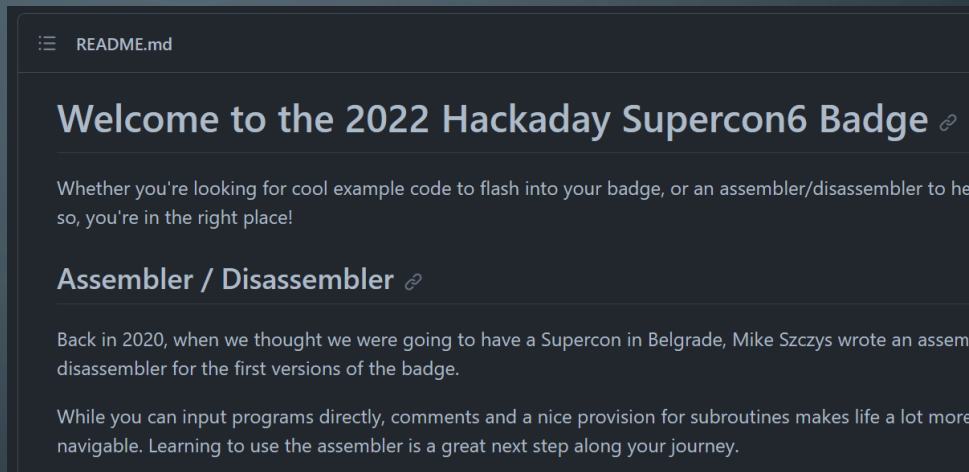
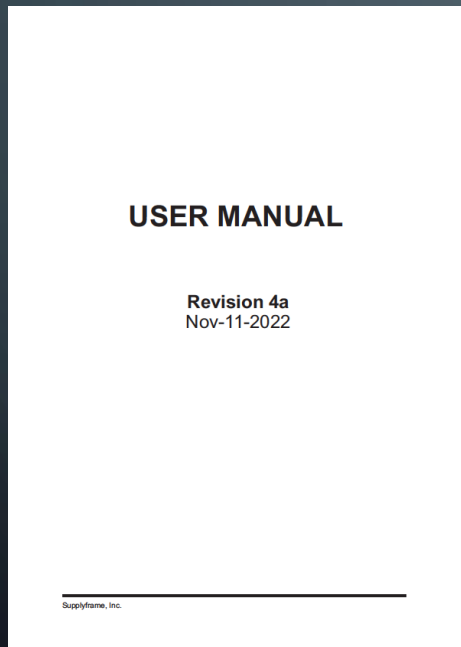
Fun!

The background is a dark blue gradient. In the corners, there are white line art illustrations of circuit boards. These include vertical lines, horizontal lines, and diagonal lines connecting to small circles, resembling electronic components or connectors. The lines are thin and white, contrasting with the dark background.

LET'S BUILD A COMPUTER!

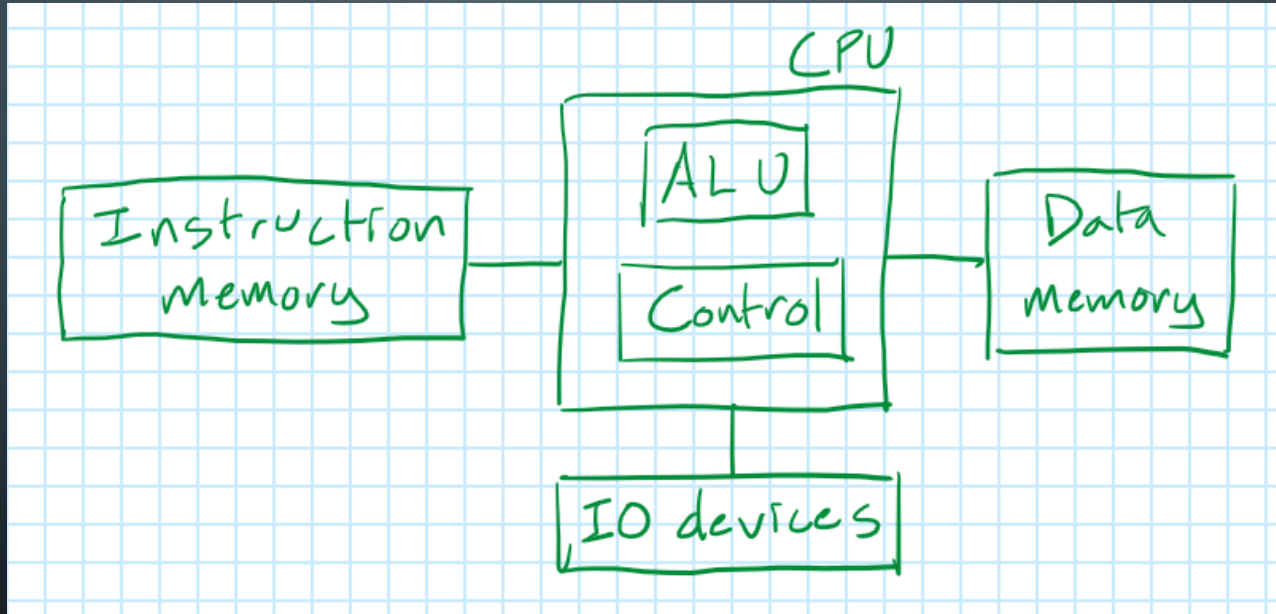
# VOJA4 RESOURCES

<https://github.com/Hack-a-Day/2022-Supercon6-Badge-Tools>

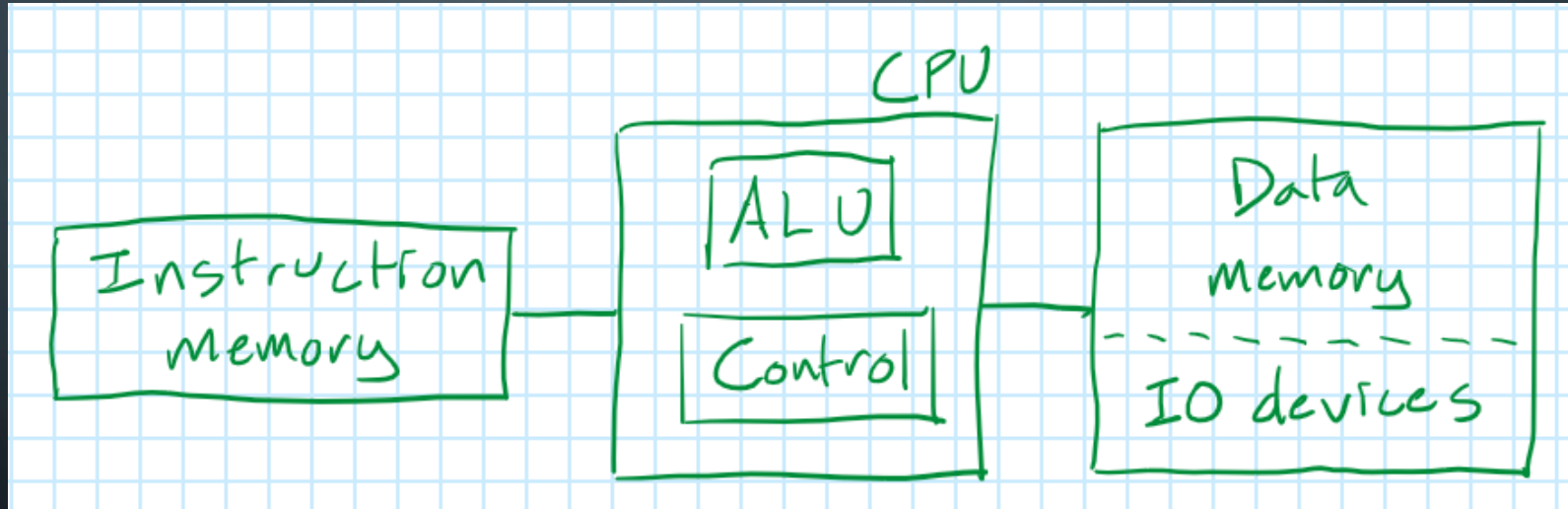


<https://hackaday.io/project/182568-badge-for-supercon6-november-2022>

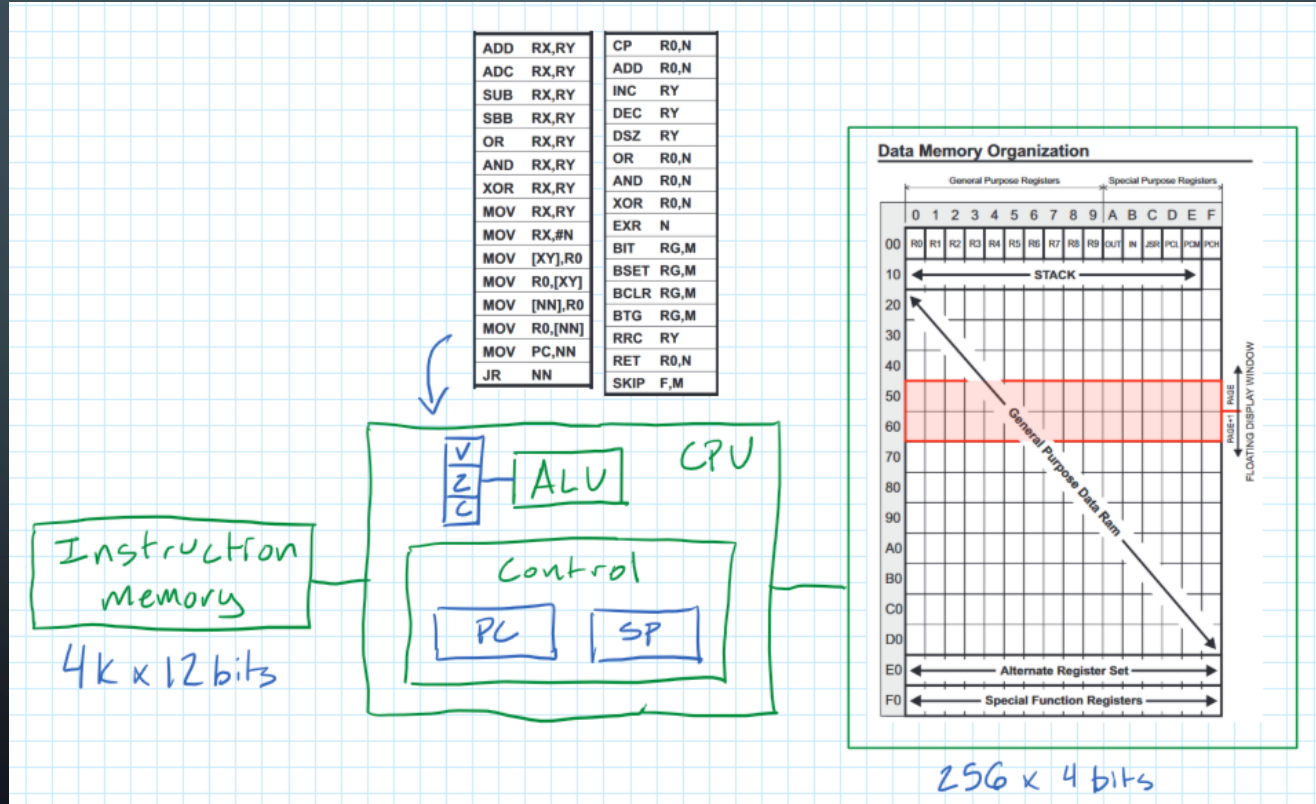
# THE VOJA4 HAS A HARVARD ARCHITECTURE



# VOJA4 USES MEMORY-MAPPED IO



# THE VOJA4 INSTRUCTION SET ARCHITECTURE (ISA)

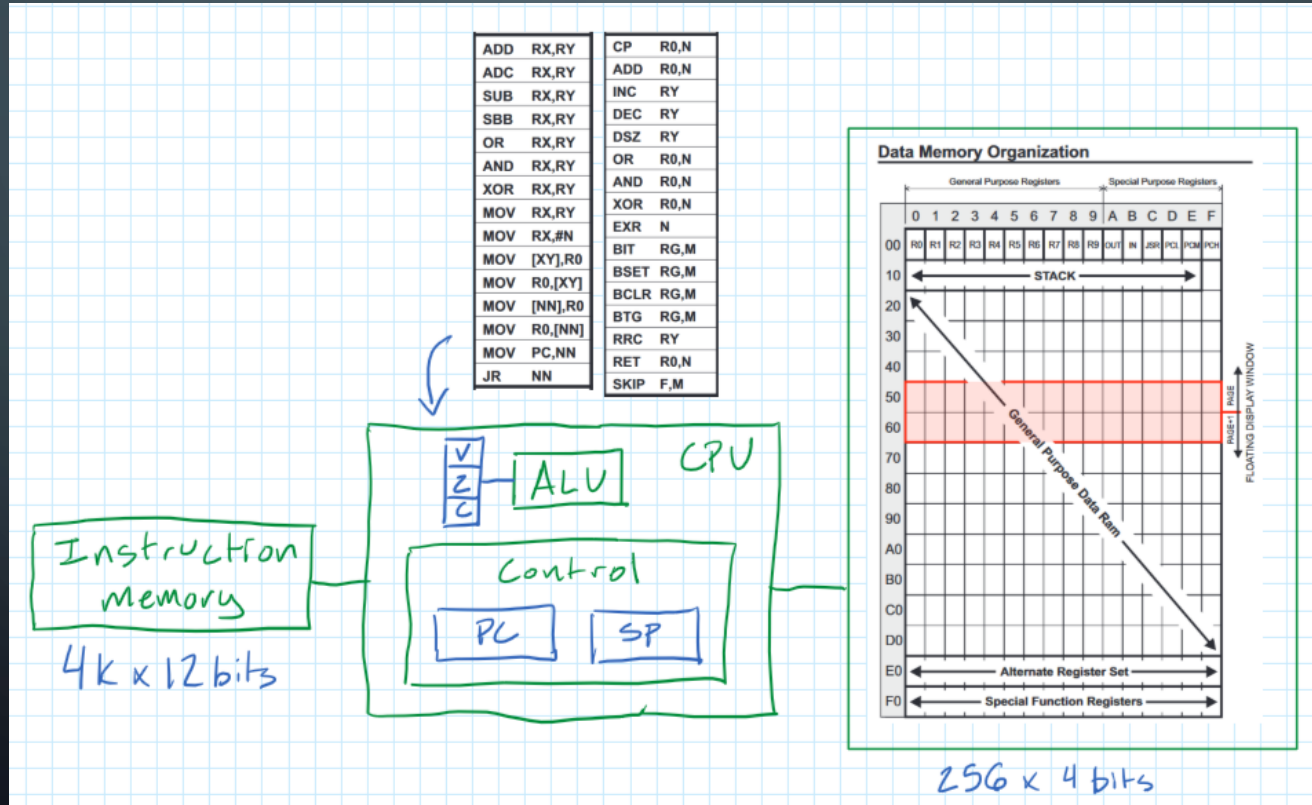


# INSTRUCTION CYCLE

Fetch

Decode

Execute

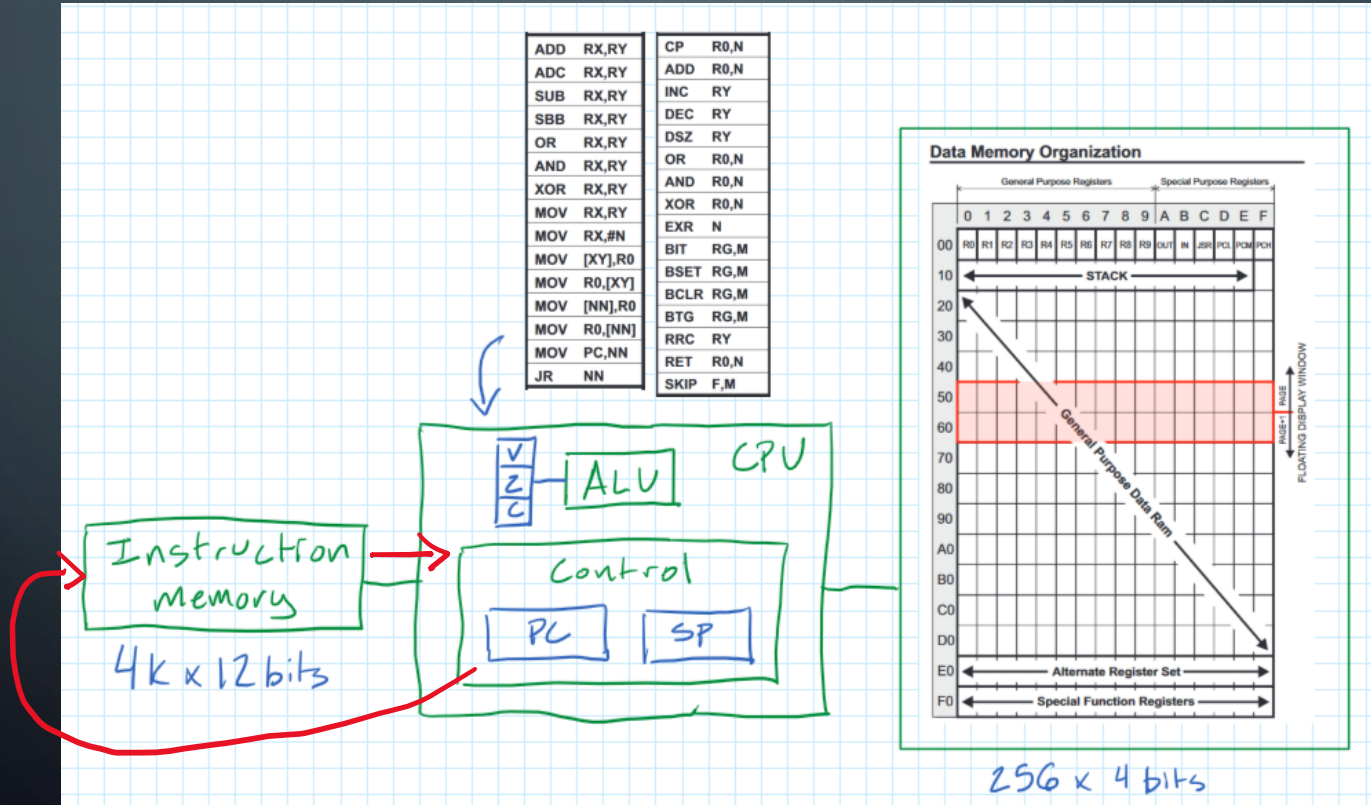


# INSTRUCTION CYCLE

Fetch

Decode

Execute

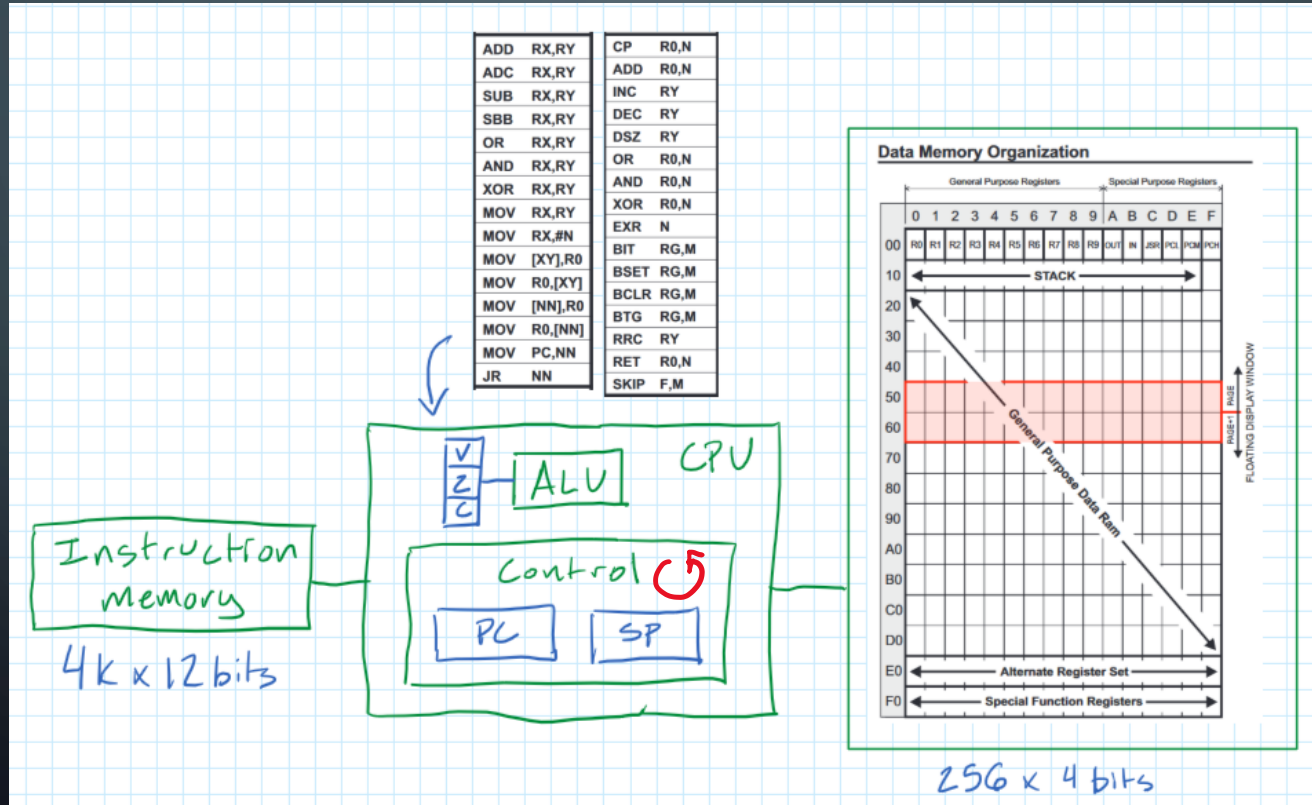


# INSTRUCTION CYCLE

Fetch

Decode

Execute

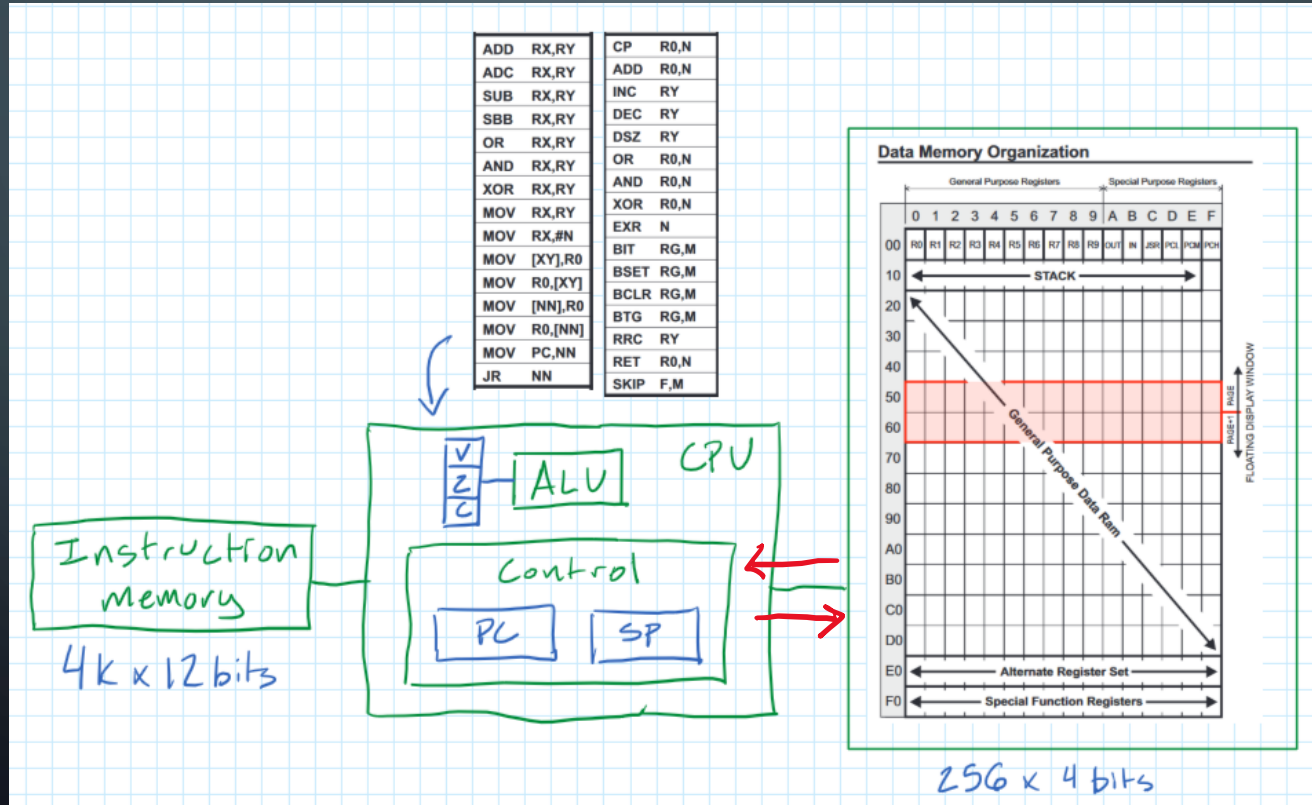


# INSTRUCTION CYCLE

Fetch

Decode

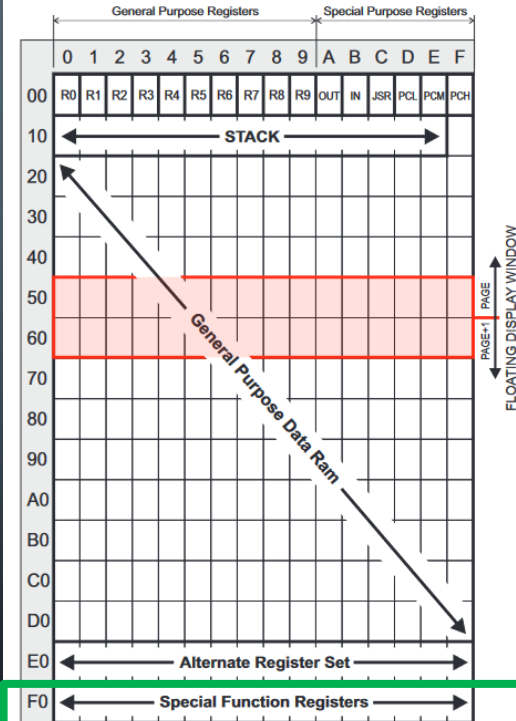
Execute



# SFRS ARE THE LAST PAGE IN DATA MEMORY

|    | NAME      | bit 3    | bit 2     | bit 1     | bit 0     | DEF  |
|----|-----------|----------|-----------|-----------|-----------|------|
| F0 | Page      |          |           |           |           | 0000 |
| F1 | Clock     |          |           |           |           | 0000 |
| F2 | Sync      |          |           |           |           | 0000 |
| F3 | WrFlags   | LedsOff  | MatrixOff | InOutPos  | RxTxPos   | 0000 |
| F4 | RdFlags   | Reserved | Reserved  | Vflag     | UserSync  | 0000 |
| F5 | SerCtrl   | RxError  |           |           |           | 0011 |
| F6 | SerLow    |          |           |           |           | 0000 |
| F7 | SerHigh   |          |           |           |           | 0000 |
| F8 | Received  |          |           |           |           | 0000 |
| F9 | AutoOff   |          |           |           |           | 0010 |
| FA | OutB      |          |           |           |           | 0000 |
| FB | InB       |          |           |           |           | 0000 |
| FC | KeyStatus | AltPress | AnyPress  | LastPress | JustPress | 0000 |
| FD | KeyReg    |          |           |           |           | 0000 |
| FE | Dimmer    |          |           |           |           | 0000 |
| FF | Random    |          |           |           |           | 0000 |

**Data Memory Organization**



# THREE GROUPS OF SFRS

|    | NAME      | bit 3                   | bit 2                   | bit 1     | bit 0     | DEF  |
|----|-----------|-------------------------|-------------------------|-----------|-----------|------|
| F0 | Page      | <div>← PAGE →</div>     |                         |           |           | 0000 |
| F1 | Clock     | <div>← CLOCK →</div>    |                         |           |           | 0000 |
| F2 | Sync      | <div>← SYNC →</div>     |                         |           |           | 0000 |
| F3 | WrFlags   | LedsOff                 | MatrixOff               | InOutPos  | RxTxPos   | 0000 |
| F4 | RdFlags   | Reserved                | Reserved                | Vflag     | UserSync  | 0000 |
| F5 | SerCtrl   | RxError                 | <div>← BaudRate →</div> |           |           | 0011 |
| F6 | SerLow    | <div>← SER LOW →</div>  |                         |           |           | 0000 |
| F7 | SerHigh   | <div>← SER HIGH →</div> |                         |           |           | 0000 |
| F8 | Received  | <div>← RECEIVED →</div> |                         |           |           | 0000 |
| F9 | AutoOff   | <div>← AUTO OFF →</div> |                         |           |           | 0010 |
| FA | OutB      | <div>← OUT B →</div>    |                         |           |           | 0000 |
| FB | InB       | <div>← IN B →</div>     |                         |           |           | 0000 |
| FC | KeyStatus | AltPress                | AnyPress                | LastPress | JustPress | 0000 |
| FD | KeyReg    | <div>← KEY REG →</div>  |                         |           |           | 0000 |
| FE | Dimmer    | <div>← DIMMER →</div>   |                         |           |           | 0000 |
| FF | Random    | <div>← RANDOM →</div>   |                         |           |           | 0000 |

## Control the processor:

- Clock
- WrFlags.InOutPos
- WrFlags.RxTxPos
- RdFlags.Vflag

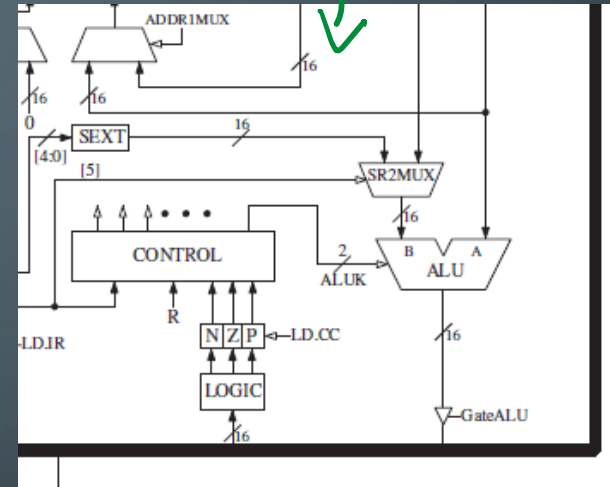
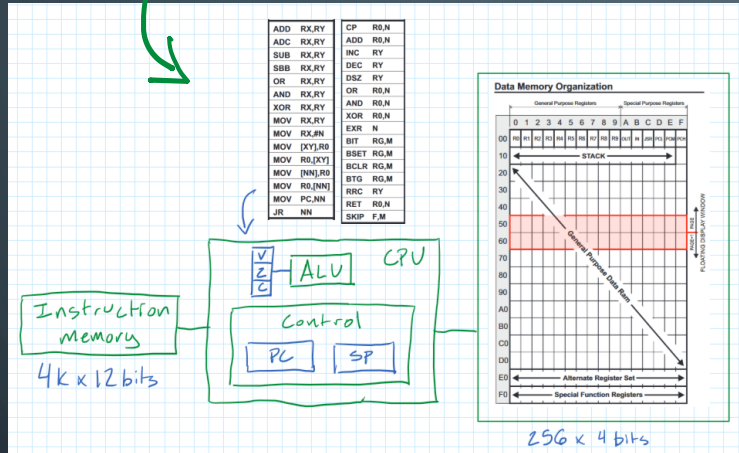
## Control the LEDs:

- Page
- AutoOff
- Dimmer
- WrFlags.LedsOff
- WrFlags.MatrixOff

## Peripherals:

- Timer
  - Sync
  - RdFlags.UserSync
- UART
  - SerCtrl
  - SerLow
  - SerHigh
  - Received
- GPIO
  - OutB
  - InB
- Keypad
  - KeyStatus
  - KeyReg
- PRNG
  - Random

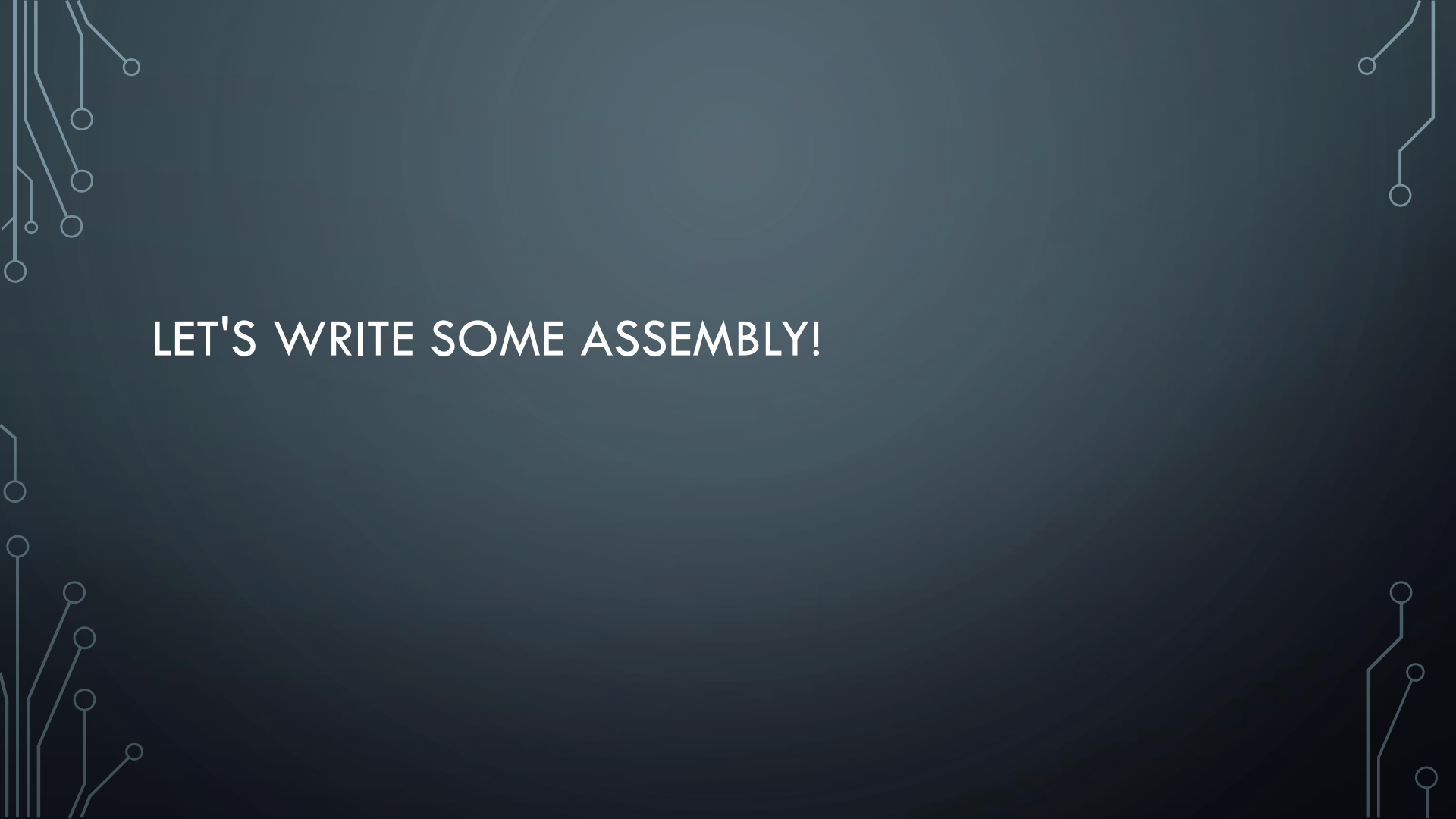
# ARCHITECTURE VS ORGANIZATION



“Circular buffer”

```
int buffer[10] = {0};
int head = 0, tail = 0;

void write(int val){
    if !isFull(){
        buffer[head] = val;
        head = (head+1)%10;
    }
}
```

The image features a dark blue gradient background with faint, stylized circuit board traces in the corners. These traces are composed of thin white lines and small circles, resembling electronic components or connectors. They are located in the top-left, top-right, bottom-left, and bottom-right corners, framing the central text.

LET'S WRITE SOME ASSEMBLY!



THREE  
CATEGORIES  
OF ASSEMBLY  
INSTRUCTIONS

---

Arithmetic and logical

---

Data movement

---

Control and branching

# ALU OPERATIONS

## 2 operand instructions

**BSET** RG,M

**BCLR** RG,M

**BTG** RG,M

**ADD** RX,RY

**ADC** RX,RY

**SUB** RX,RY

**SBB** RX,RY

**OR** RX,RY

**AND** RX,RY

**XOR** RX,RY

## 1 operand instructions

**ADD** R0,N

**OR** R0,N

**AND** R0,N

**XOR** R0,N

**INC** RY

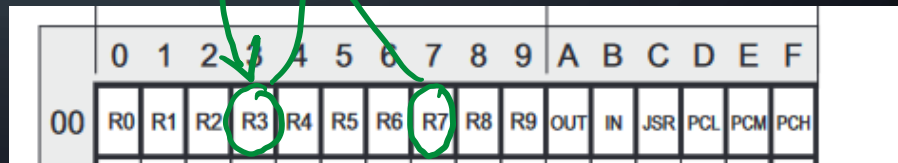
**DEC** RY

**DSZ** <sup>\*</sup>RY

**RRC** RY

Ex:

```
ADD R3, R7      ; R3 <-- R3 + R7
                  ;           R3      R7
                  ; Before:   5       8
                  ; After:   13      8
```



# EMULATED OR "SYNTHETIC" ALU OPERATIONS

| Non-existing instruction |                                                         | Replace with                                          |                                                     |
|--------------------------|---------------------------------------------------------|-------------------------------------------------------|-----------------------------------------------------|
| <b>RLC</b>               | <b>RX, RY</b> Rotate Left <b>RY</b> through Carry       | <b>MOV</b> <b>RX,RY</b><br><b>ADDC</b> <b>RX, RY</b>  | Note: Result is in <b>RX</b>                        |
| <b>SL</b>                | <b>RX, RY</b> Shift Left <b>RY</b>                      | <b>MOV</b> <b>RX,RY</b><br><b>ADD</b> <b>RX, RY</b>   | Note: Result is in <b>RX</b>                        |
| <b>LSR</b>               | <b>RY</b> Logical Shift Right <b>RY</b>                 | <b>AND</b> <b>R0, 0xF</b><br><b>RRC</b> <b>RY</b>     |                                                     |
| <b>CPL</b>               | <b>R0</b> Complement <b>R0</b>                          | <b>XOR</b> <b>R0, 0xF</b>                             |                                                     |
| <b>CPL</b>               | <b>RX, RY</b> Complement <b>RY</b> , write to <b>RX</b> | <b>MOV</b> <b>RX, 0xF</b><br><b>SUB</b> <b>RX, RY</b> | Note: Result is in <b>RX</b>                        |
| <b>NEG</b>               | <b>RX, RY</b> Negate <b>RY</b> , write to <b>RX</b>     | <b>MOV</b> <b>RX, 0</b><br><b>SUB</b> <b>RX, RY</b>   | Note: Result is in <b>RX</b>                        |
| <b>NOP</b>               | No Operation                                            | <b>MOV</b> <b>R0, R0</b>                              | Note: Any Register except <b>R12</b> and <b>R13</b> |

# SIMPLE FIBONACCI PROGRAM

Big-endian

16-bit  
addition

```
MOV R7, 1
```

```
LOOP:
```

```
ADD R3, R7
```

```
ADC R2, R6
```

```
ADC R1, R5
```

```
ADC R0, R4
```

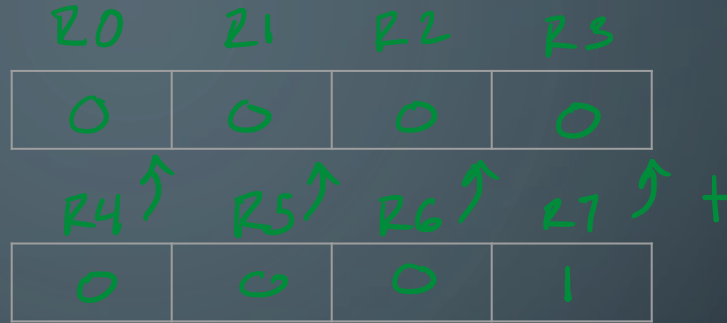
```
ADD R7, R3
```

```
ADC R6, R2
```

```
ADC R5, R1
```

```
ADC R4, R0
```

```
JR LOOP
```



# SIMPLE FIBONACCI PROGRAM

Little-endian

16-bit  
addition

MOV R4, 1

LOOP:

ADD R0, R4

ADC R1, R5

ADC R2, R6

ADC R3, R7

ADD R4, R0

ADC R5, R1

ADC R6, R2

ADC R7, R3

JR LOOP

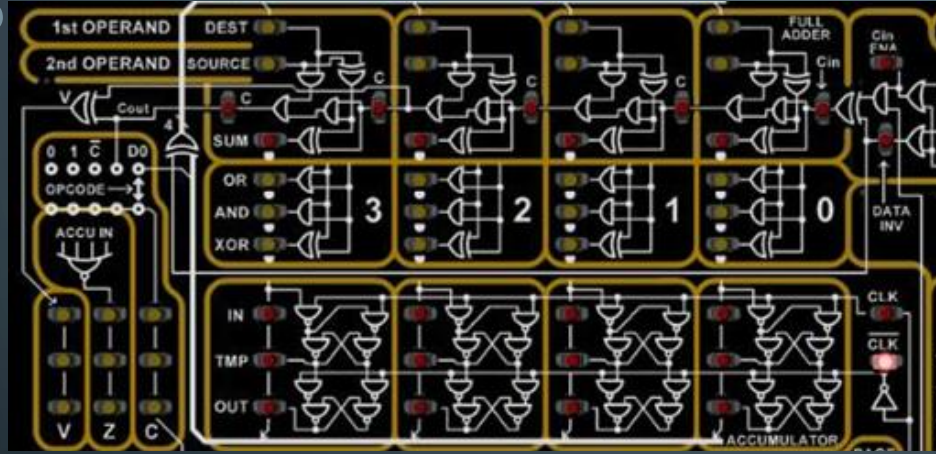
| R3 | R2 | R1 | R0 |
|----|----|----|----|
| 0  | 0  | 0  | 0  |

| R7 | R6 | R5 | R4 |
|----|----|----|----|
| 0  | 0  | 0  | 1  |

| R3 | R2 | R1 | R0 |
|----|----|----|----|
| 0  | 0  | 0  | 1  |

| R7 | R6 | R5 | R4 |
|----|----|----|----|
| 0  | 0  | 0  | 1  |

# DIR MODE



Supports:

**ADD RX,RX**

**ADC RX,RX**

**SUB RX,RX**

**SBB RX,RX**

**OR RX,RX**

**AND RX,RX**

**XOR RX,RX**

**MOV RX,RX**

**INC RY**

**DEC RY**

**RRC RY**

| MODE    | COMMAND KEYS      |                                       |                                         |                     | OPCODE                         | OPER X                                | OPER Y                               |
|---------|-------------------|---------------------------------------|-----------------------------------------|---------------------|--------------------------------|---------------------------------------|--------------------------------------|
| ALT DIR | Carry             | Save                                  | Load                                    | Clock               | Opcode                         | Oper X                                | Oper Y                               |
|         | Toggle Carry Flag | Send Program Memory to Serial Port    | Load Program Memory from Serial Port    | Master Clock source | Instruction Opcode (bits 11-8) | Direct Operand X or Opcode (bits 7-4) | Direct Operand Y (bits 3-0)          |
|         |                   | Save Program Memory to selected Flash | Load Program Memory from selected Flash |                     | Dimmer level select            | Baud Rate select                      | Flash portion select for Save / Load |

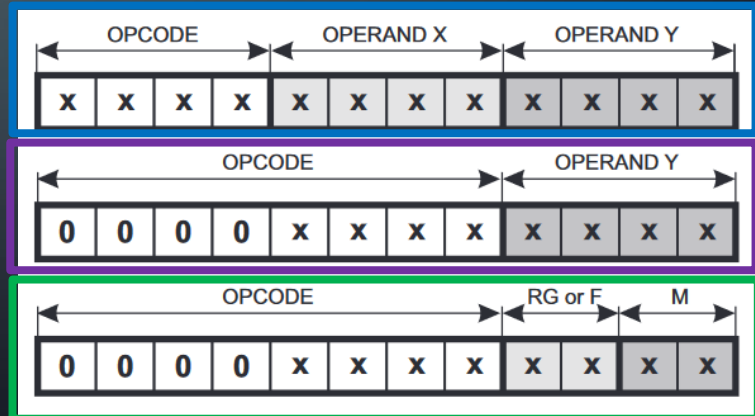
## 2 operand instructions

|      |      |
|------|------|
| BSET | RG,M |
| BCLR | RG,M |
| BTG  | RG,M |

|            |              |
|------------|--------------|
| <b>ADD</b> | <b>RX,RY</b> |
| <b>ADC</b> | <b>RX,RY</b> |
| <b>SUB</b> | <b>RX,RY</b> |
| <b>SBB</b> | <b>RX,RY</b> |
| <b>OR</b>  | <b>RX,RY</b> |
| <b>AND</b> | <b>RX,RY</b> |
| <b>XOR</b> | <b>RX,RY</b> |

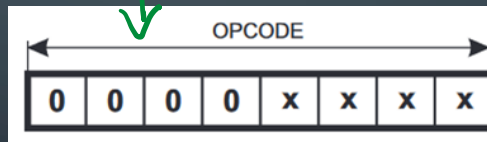
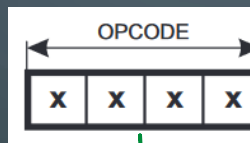
## 1 operand instructions

|     |      |     |    |
|-----|------|-----|----|
| ADD | R0,N | INC | RY |
| OR  | R0,N | DEC | RY |
| AND | R0,N | DSZ | RY |
| XOR | R0,N | RRC | RY |



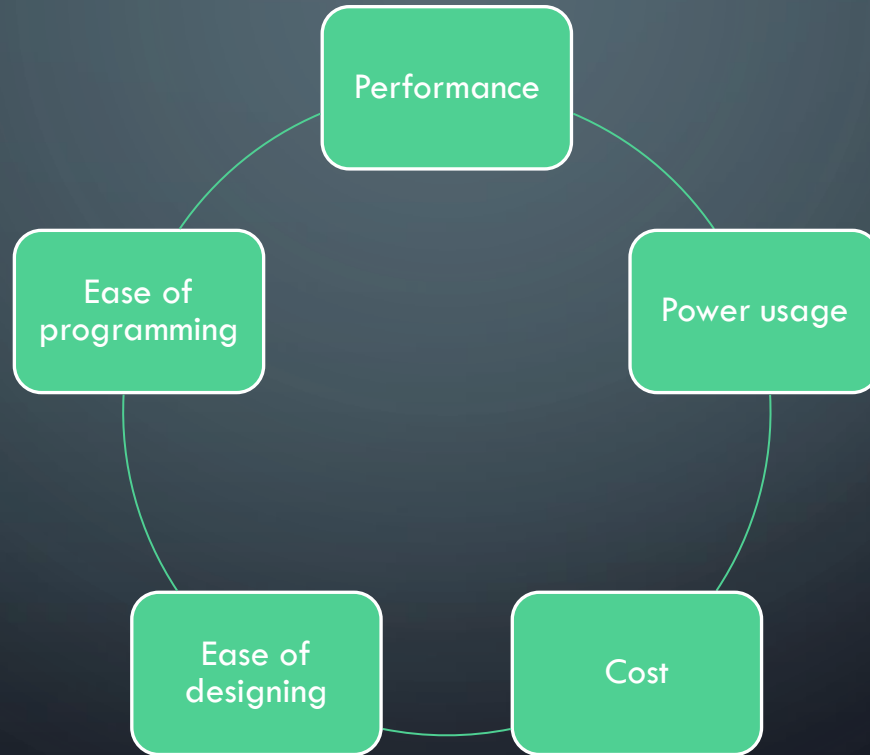
# EXPANDING OPCODES

|     |         |
|-----|---------|
| ADD | RX,RY   |
| ADC | RX,RY   |
| SUB | RX,RY   |
| SBB | RX,RY   |
| OR  | RX,RY   |
| AND | RX,RY   |
| XOR | RX,RY   |
| MOV | RX,RY   |
| MOV | RX,#N   |
| MOV | [XY],R0 |
| MOV | R0,[XY] |
| MOV | [NN],R0 |
| MOV | R0,[NN] |
| MOV | PC,NN   |
| JR  | NN      |



|      |      |
|------|------|
| CP   | R0,N |
| ADD  | R0,N |
| INC  | RY   |
| DEC  | RY   |
| DSZ  | RY   |
| OR   | R0,N |
| AND  | R0,N |
| XOR  | R0,N |
| EXR  | N    |
| BIT  | RG,M |
| BSET | RG,M |
| BCLR | RG,M |
| BTG  | RG,M |
| RRC  | RY   |
| RET  | R0,N |
| SKIP | F,M  |

# DESIGN TRADE-OFFS



# DESIGN TRADE-OFFS

①

|     |
|-----|
| ADD |
| ADC |
| SUB |
| SBB |
| OR  |
| AND |
| XOR |

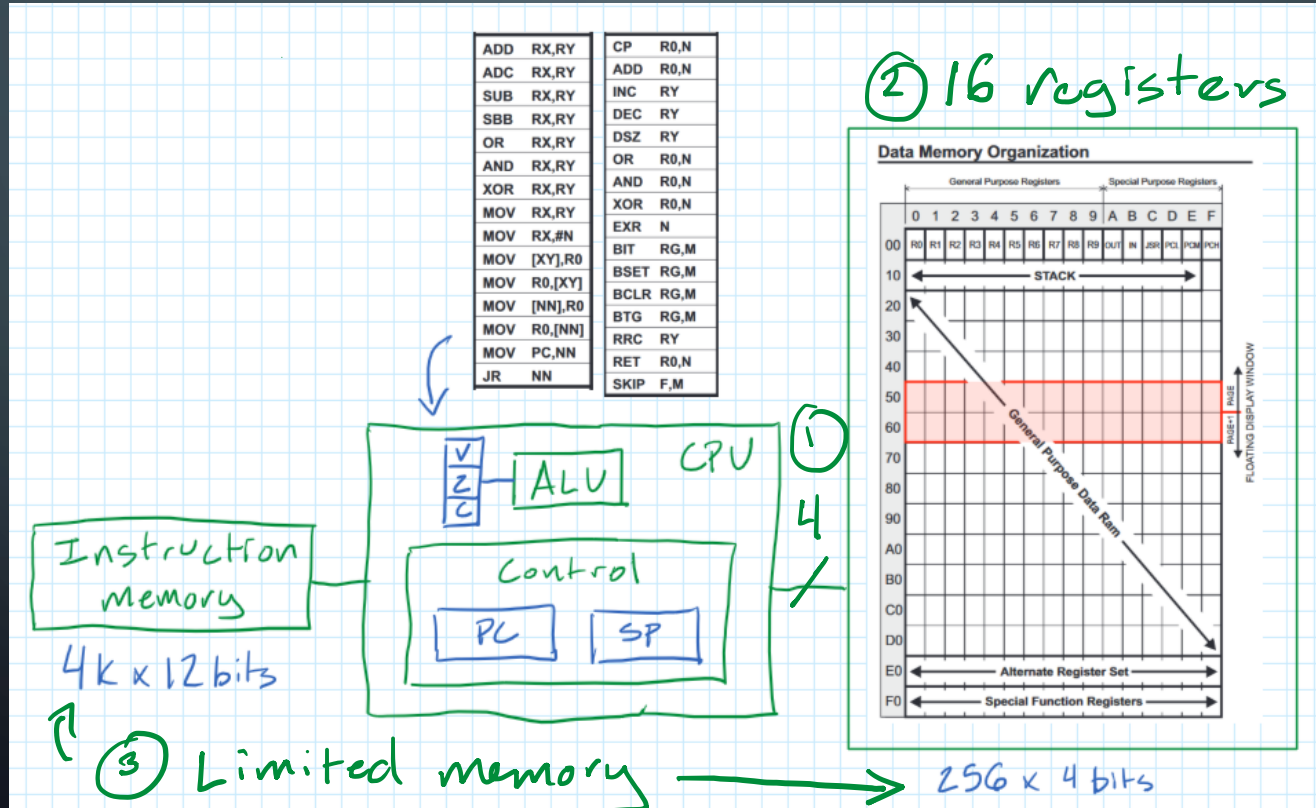


② 4-bit operands

④ Fixed-length

③ 2 registers/values

# DESIGN TRADE-OFFS



# DATA MOVEMENT OPERATIONS

|              |                |
|--------------|----------------|
| <b>MOV</b>   | <b>RX,RY</b>   |
| <b>MOV</b>   | <b>RX,#N</b>   |
| <b>MOV</b>   | <b>[XY],R0</b> |
| <b>MOV</b>   | <b>R0,[XY]</b> |
| <b>MOV</b>   | <b>[NN],R0</b> |
| <b>MOV</b>   | <b>R0,[NN]</b> |
| <b>MOV</b>   | <b>PC,NN</b>   |
| <b>EXR</b>   | <b>N</b>       |
| <b>RET</b> * | <b>R0,N</b>    |

Ex:

MOV R1, R5 ; R1 <-- R5

; R1 R5

; Before: 10 3

; After: 3 3

|    | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | A   | B  | C   | D   | E   | F   |
|----|----|----|----|----|----|----|----|----|----|----|-----|----|-----|-----|-----|-----|
| 00 | R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7 | R8 | R9 | OUT | IN | JSR | PCL | PCM | PCH |

Ex:

MOV PC, 0x0F ; PCH <-- 0x0, PCM <-- 0xF

# DATA MOVEMENT OPERATIONS

## Pointers

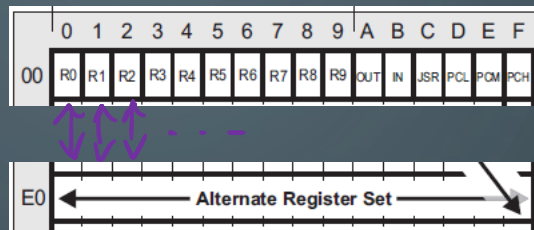
|              |                |
|--------------|----------------|
| <b>MOV</b>   | <b>RX,RY</b>   |
| <b>MOV</b>   | <b>RX,#N</b>   |
| <b>MOV</b>   | <b>[XY],R0</b> |
| <b>MOV</b>   | <b>R0,[XY]</b> |
| <b>MOV</b>   | <b>[NN],R0</b> |
| <b>MOV</b>   | <b>R0,[NN]</b> |
| <b>MOV</b>   | <b>PC,NN</b>   |
| <b>EXR</b>   | <b>N</b>       |
| <b>RET</b> * | <b>R0,N</b>    |

|                                |                                                                                                                                              |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
|                                | <pre>; We have to manage memory<br/>directly with the Voja4, since<br/>there is no compiler.<br/><br/>; Assume we assigned "a" to 0x3F</pre> |
| <pre>int a = 5;</pre>          | <pre>MOV R0, 5<br/>MOV [0x3F], R0</pre>                                                                                                      |
| <pre>int * p_a = &amp;a;</pre> | <pre>MOV R2, 0x3<br/>MOV R3, 0xF</pre>                                                                                                       |
| <pre>*p_a = 6;</pre>           | <pre>MOV R0, 6<br/>MOV [R2:R3], R0</pre>                                                                                                     |

# DATA MOVEMENT OPERATIONS

|     |         |
|-----|---------|
| MOV | RX,RY   |
| MOV | RX,#N   |
| MOV | [XY],R0 |
| MOV | R0,[XY] |
| MOV | [NN],R0 |
| MOV | R0,[NN] |
| MOV | PC,NN   |
| EXR | N       |
| RET | * R0,N  |

## Alternate registers



NOT for saving  
registers in a  
subroutine

## Const data

```
MOV PC, [HIGH Table: MID Table]
MOV JSR, LOW Table      ; R0 gets 0xC (Table[0])
INC JSR                  ; R0 gets 0x0 (Table[1])
INC JSR                  ; R0 gets 0xD (Table[2])
INC JSR                  ; R0 gets 0xE (Table[3])
```

Table:

```
RET R0, 0xC
RET R0, 0x0
RET R0, 0xD
RET R0, 0xE
```

EQU (assembler directive, like #define)  
min EQU 10  
CP R0, min

# ADDRESSING MODES

|                                                   |                                                                      |                                                                                                |
|---------------------------------------------------|----------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| <b>Immediate</b><br>Operand = Value<br>MOV R0, 4  | <b>PC Relative</b><br>Operand = PC + Value<br>JR -1                  | <b>Memory Absolute</b><br>Operand = M[Value]<br><del>MOV R0, [0x3E]</del><br>MOV [0x0], [0x3E] |
| <b>Implicit</b><br>Operand in opcode<br>MOV R0, 4 | <b>Memory Indirect</b><br>Operand = M[M[Value]]<br>MOV R0, [R8 : R9] |                                                                                                |

## Others??

- “Memory indirect indirect”: Operand = M[M[M[value]]]
- “Memory offset”: Operand = M[M[Value] + offset]
- “Base plus index”: Operand = M[M[Value] + M[Value]]

# DATA MOVEMENT IN "TERTIS" [KOPpanyh]

```
translate:
mov r3, 0xB
mov r4, 0
mov r0, [r3:r4]
add r0, r1
mov [r3:r4], r0
inc r4
mov r0, [r3:r4]
add r0, r2
mov [r3:r4], r0
inc r4
mov r0, r4
cp r0, 8
skip z, 1
jr -12
gosub normalize
ret r0, 0
```

$x += r1$

$y += r2$

Ptr to xy  
coords in  
mem



|    |    |     |    |
|----|----|-----|----|
| B0 | B1 | ... | B7 |
| x1 | y1 |     | y4 |

# CONTROL OPERATIONS

Change the PC

|                                                                                              |            |
|----------------------------------------------------------------------------------------------|------------|
| <b>JR</b>                                                                                    | <b>NN</b>  |
| <b>SKIP</b>                                                                                  | <b>F,M</b> |
| <b>DSZ</b>  | <b>RY</b>  |

Writing to PCL

Change the PC + SP

|            |             |
|------------|-------------|
| <b>RET</b> | <b>R0,N</b> |
|------------|-------------|

Writing to JSR

Change the ALU flags

|            |             |
|------------|-------------|
| <b>CP</b>  | <b>R0,N</b> |
| <b>BIT</b> | <b>RG,M</b> |

Writing to RdFlags.Vflag  
Also ADD, ADC, SUB, SBB,  
OR, AND, XOR, ADD, INC,  
DEC, RRC

# CONTROL OPERATIONS

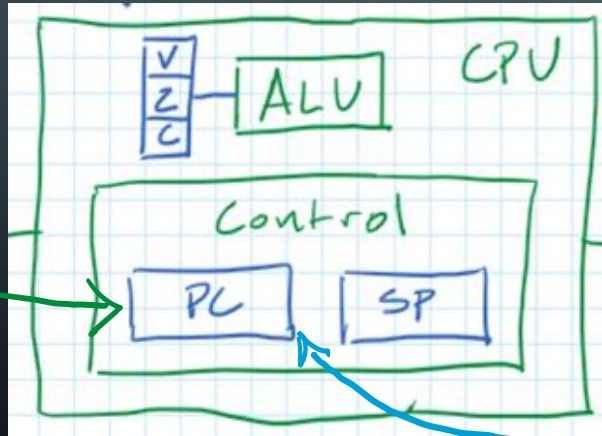
Change the PC

JR NN

SKIP F,M

DSZ <sup>\*</sup>RY

Writing to PCL



## Conditional branches

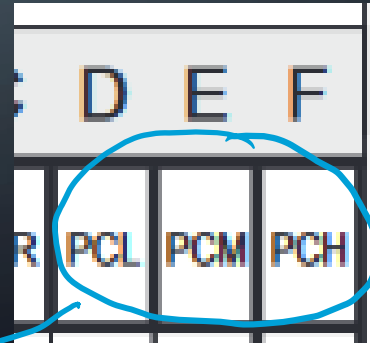
SKIP/DSZ [Optional: + JR/MOV]

SKIP C, 1 ; Jump to LOOP if Z==0  
JR LOOP

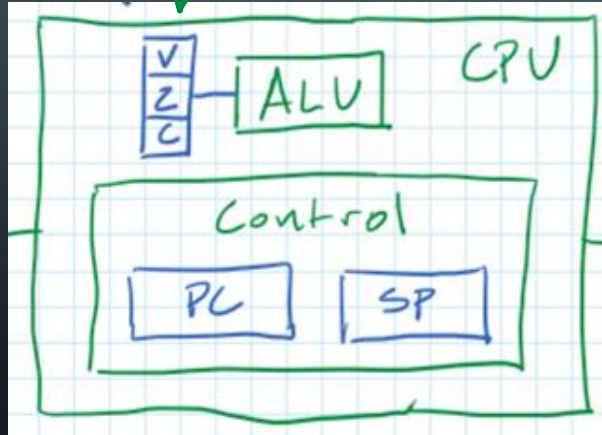
DSZ R4 ; Call subroutine if R4!=0  
MOV JSR, LOW SubroutineA

## Long jumps

MOV PC, [High A : Mid A]  
MOV PCL, Low A



# CONTROL OPERATIONS



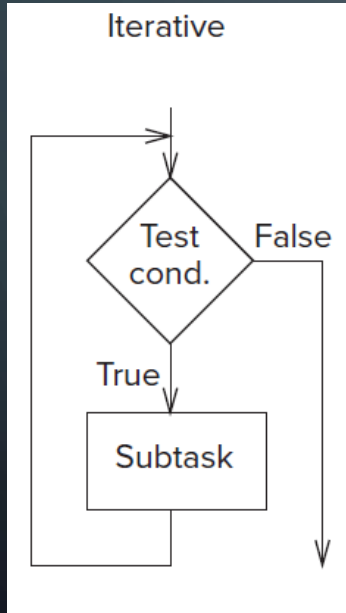
Change the ALU flags

|    |      |
|----|------|
| CP | R0,N |
|----|------|

|     |      |
|-----|------|
| BIT | RG,M |
|-----|------|

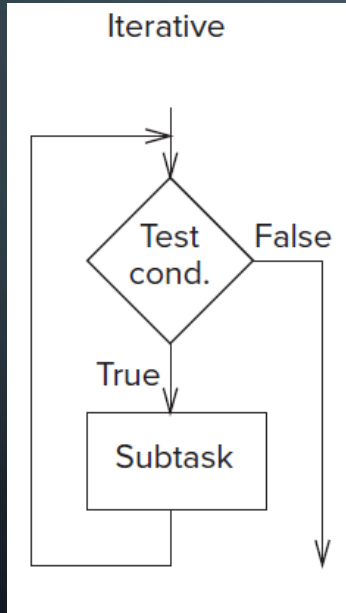
Writing to RdFlags.Vflag  
Also ADD, ADC, SUB, SBB,  
OR, AND, XOR, ADD, INC,  
DEC, RRC

# WRITING A LOOP IN ASSEMBLY



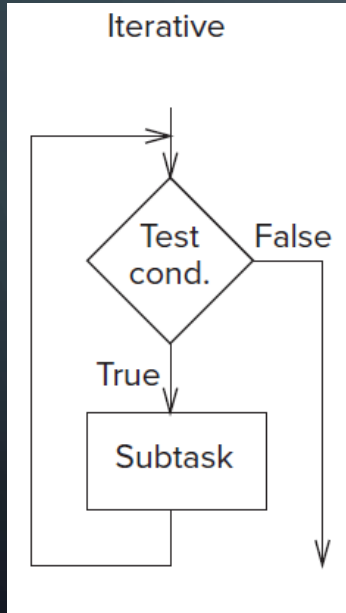
|       |                                                                  |
|-------|------------------------------------------------------------------|
| Setup | <pre>; for(int idx = 15; idx &gt; 0; idx--)<br/>MOV R0, 15</pre> |
|       |                                                                  |
|       |                                                                  |
|       |                                                                  |
|       |                                                                  |
|       |                                                                  |

# WRITING A LOOP IN ASSEMBLY



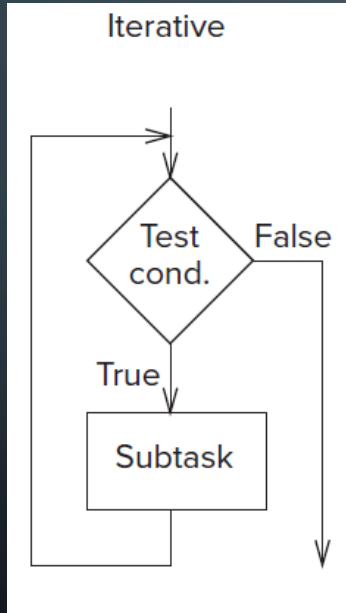
|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| Setup                   | <pre>; for(int idx = 15; idx &gt; 0; idx--)<br/>MOV R0, 15</pre> |
| Generate test condition | <pre>CP R0, 0</pre>                                              |
|                         |                                                                  |
|                         |                                                                  |
|                         |                                                                  |
|                         |                                                                  |

# WRITING A LOOP IN ASSEMBLY



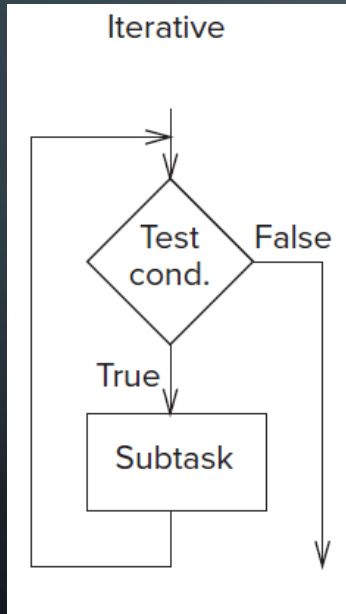
|                          |                                                                  |
|--------------------------|------------------------------------------------------------------|
| Setup                    | <pre>; for(int idx = 15; idx &gt; 0; idx--)<br/>MOV R0, 15</pre> |
| Generate test condition  | <pre>CP R0, 0</pre>                                              |
| Jump below loop if false | <pre>SKIP NZ, 1<br/>JR DONE</pre>                                |
|                          |                                                                  |
|                          |                                                                  |
|                          | <pre>DONE:<br/>; Code that comes after the loop...</pre>         |

# WRITING A LOOP IN ASSEMBLY



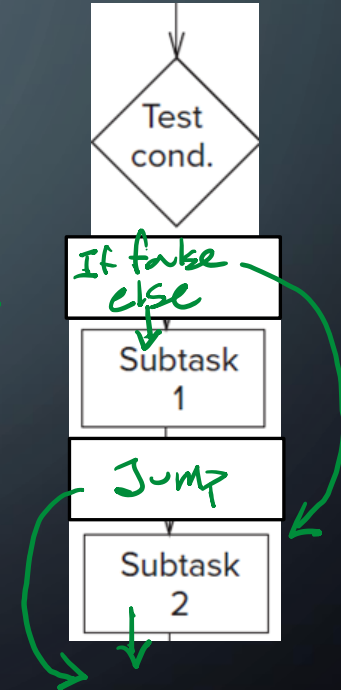
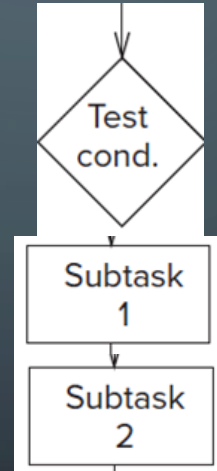
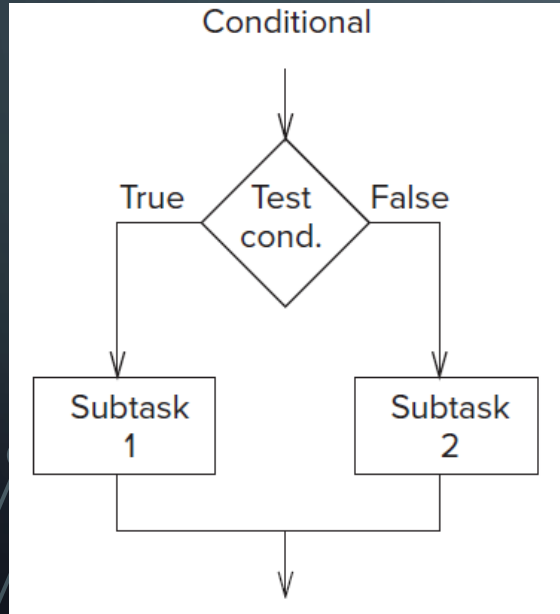
|                          |                                                                  |
|--------------------------|------------------------------------------------------------------|
| Setup                    | <pre>; for(int idx = 15; idx &gt; 0; idx--)<br/>MOV R0, 15</pre> |
| Generate test condition  | <pre>CP R0, 0</pre>                                              |
| Jump below loop if false | <pre>SKIP NZ, 1<br/>JR DONE</pre>                                |
| Execute loop body        | <pre>; Loop body<br/>DEC R0</pre>                                |
|                          |                                                                  |
|                          | <pre>DONE:<br/>; Code that comes after the loop...</pre>         |

# WRITING A LOOP IN ASSEMBLY



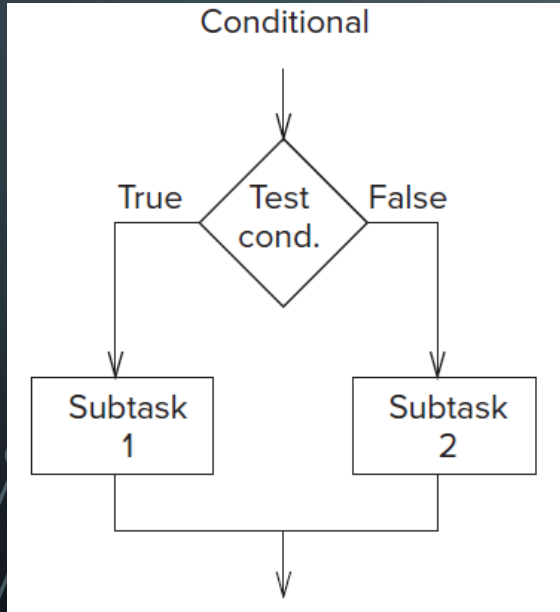
|                                   |                                                                  |
|-----------------------------------|------------------------------------------------------------------|
| Setup                             | <pre>; for(int idx = 15; idx &gt; 0; idx--)<br/>MOV R0, 15</pre> |
| Generate test condition           | <pre>LOOP:<br/>CP R0, 0</pre>                                    |
| Jump below loop if false          | <pre>SKIP NZ, 1<br/>JR DONE</pre>                                |
| Execute loop body                 | <pre>; Loop body<br/>DEC R0</pre>                                |
| Unconditional jump to top of loop | <pre>JR LOOP</pre>                                               |
|                                   | <pre>DONE:<br/>; Code that comes after the loop...</pre>         |

# WRITING A CONDITIONAL IN ASSEMBLY



# WRITING A CONDITIONAL IN ASSEMBLY

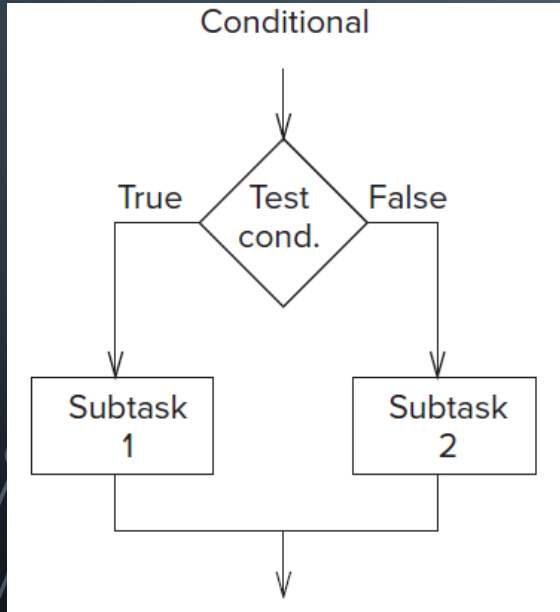
```
if R0 == 0
    // "if" block
else
    // "else block"
```



|                         |          |
|-------------------------|----------|
| Generate test condition | CP R0, 0 |
|                         |          |
|                         |          |
|                         |          |
|                         |          |
|                         |          |

# WRITING A CONDITIONAL IN ASSEMBLY

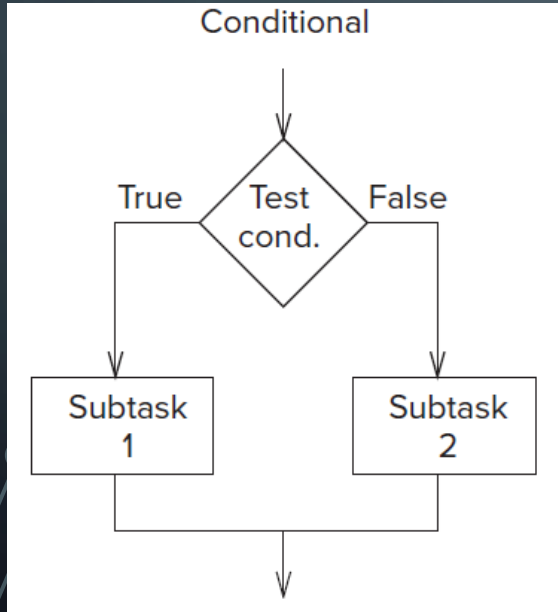
```
if R0 == 0
    // "if" block
else
    // "else block"
```



|                         |              |
|-------------------------|--------------|
| Generate test condition | CP R0, 0     |
|                         |              |
| Execute "if" block      | ; "if" block |
|                         |              |
|                         |              |
|                         |              |

# WRITING A CONDITIONAL IN ASSEMBLY

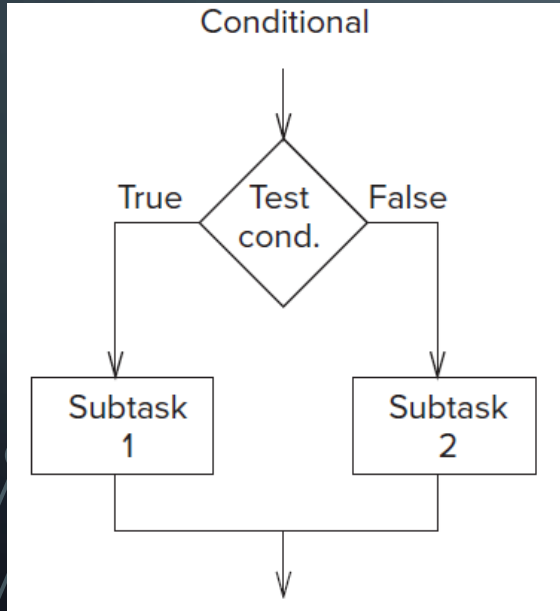
```
if R0 == 0
    // "if" block
else
    // "else block"
```



|                         |                                                        |
|-------------------------|--------------------------------------------------------|
| Generate test condition | CP R0, 0                                               |
|                         |                                                        |
| Execute "if" block      | ; "if" block                                           |
| Jump to bottom          | JR DONE                                                |
|                         |                                                        |
|                         | DONE:<br>; Code that comes after the<br>conditional... |

# WRITING A CONDITIONAL IN ASSEMBLY

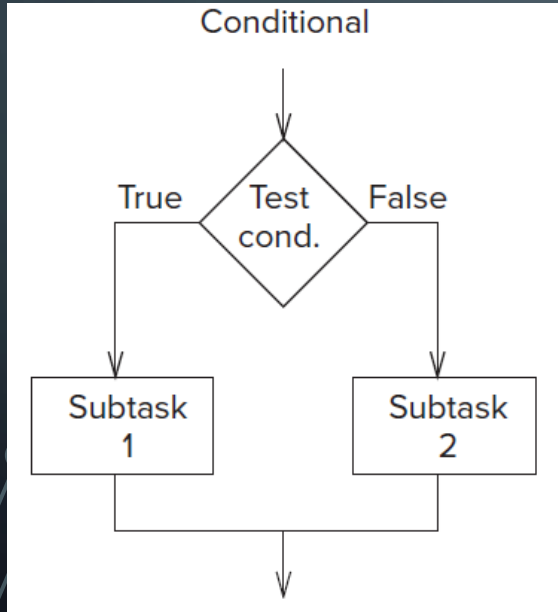
```
if R0 == 0
    // "if" block
else
    // "else block"
```



|                               |                                                     |
|-------------------------------|-----------------------------------------------------|
| Generate test condition       | CP R0, 0                                            |
| Jump to "else" block if false | SKIP NZ, 1<br>JR ELSE                               |
| Execute "if" block            | ; "if" block                                        |
| Jump to bottom                | JR DONE                                             |
|                               |                                                     |
|                               | DONE:<br>; Code that comes after the conditional... |

# WRITING A CONDITIONAL IN ASSEMBLY

```
if R0 == 0
    // "if" block
else
    // "else block"
```



|                               |                                                     |
|-------------------------------|-----------------------------------------------------|
| Generate test condition       | CP R0, 0                                            |
| Jump to "else" block if false | SKIP NZ, 1<br>JR ELSE                               |
| Execute "if" block            | ; "if" block                                        |
| Jump to bottom                | JR DONE                                             |
| Execute "else" block          | ELSE:<br>; "else" block                             |
|                               | DONE:<br>; Code that comes after the conditional... |

# CONTROL OPERATIONS

Change the PC + SP

**RET R0,N**

Writing to JSR

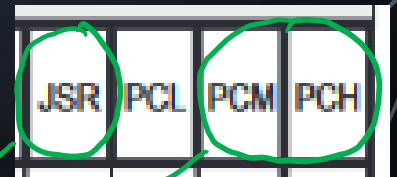
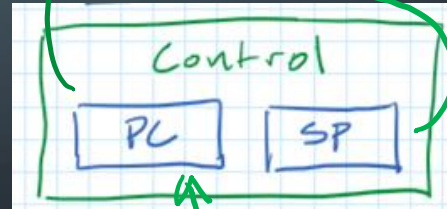
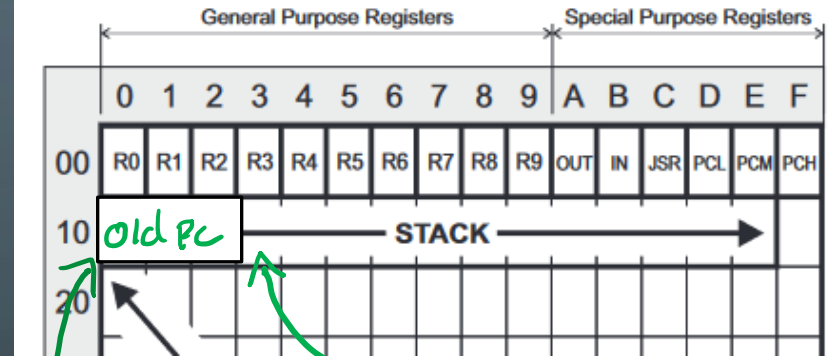
# CALLING A SUBROUTINE IN ASSEMBLY

## 1) Mechanics

```
; Set up subroutine arguments
MOV PC, [High SQR : Mid SQR]
MOV JSR, Low SQR
; (next instruction)

; SQR
; R1 = sqrt(R1)
SQR:
; Uses R2, R3 (temp registers)
; Calls subroutine (save R1)
RET R0, 0
```

## Data Memory Organization



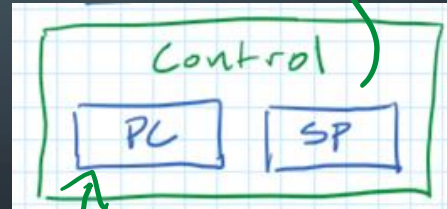
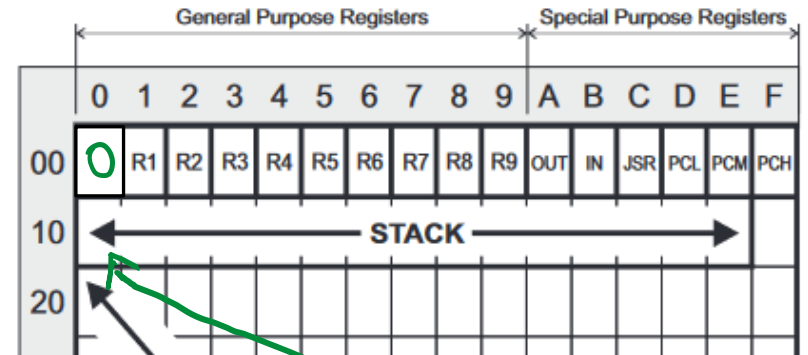
# CALLING A SUBROUTINE IN ASSEMBLY

## 1) Mechanics

```
; Set up subroutine arguments
MOV PC, [High SQRT : Mid SQRT]
MOV JSR, Low SQRT
;(next instruction)

; SQRT
; R1 = sqrt(R1)
SQRT:
; Uses R2, R3 (temp registers)
; Calls subroutine (save R1)
RET R0, 0
```

## Data Memory Organization

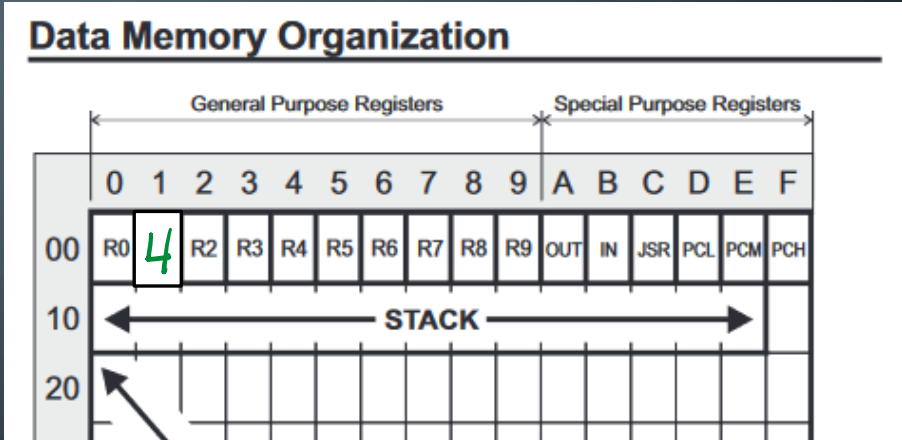


old PC

## 2) Interface/“ABI”

```
; Set up subroutine arguments
MOV PC, [High SQRT : Mid SQRT]
MOV JSR, Low SQRT
;(next instruction)

; SQRT
; R1 = sqrt(R1)
SQRT:
; Uses R2, R3 (temp registers)
; Calls subroutine (save R1)
RET R0, 0
```



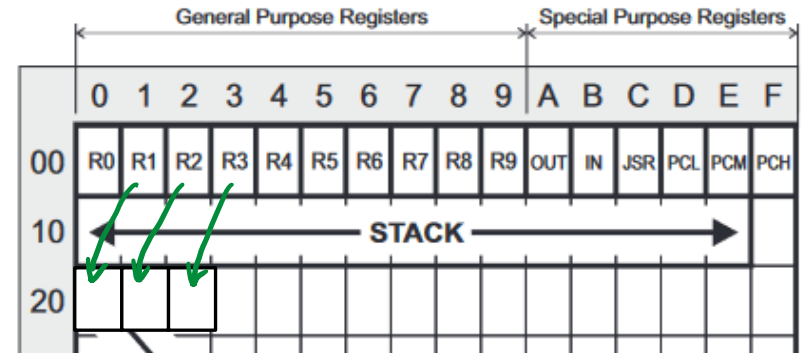
# CALLING A SUBROUTINE IN ASSEMBLY

## 3) No auto variables; globals only

```
; Set up subroutine arguments
MOV PC, [High SQRT : Mid SQRT]
MOV JSR, Low SQRT
;(next instruction)

; SQRT
; R1 = sqrt(R1)
SQRT:
; Uses R2, R3 (temp registers)
; Calls subroutine (save R1)
RET R0, 0
```

### Data Memory Organization



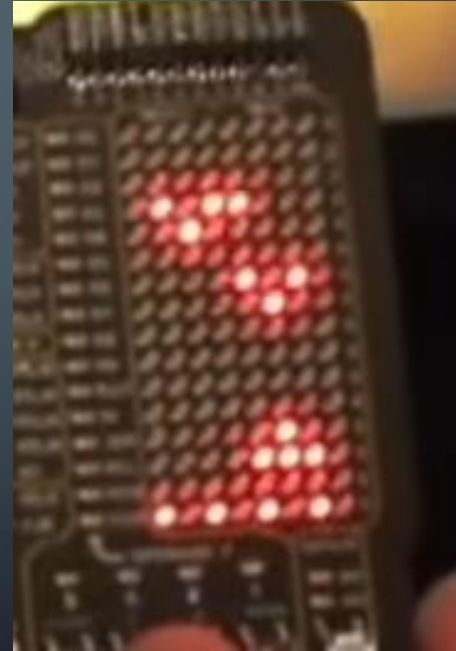
M[0x20-0x22] belong to SQRT  
(no recursion!)

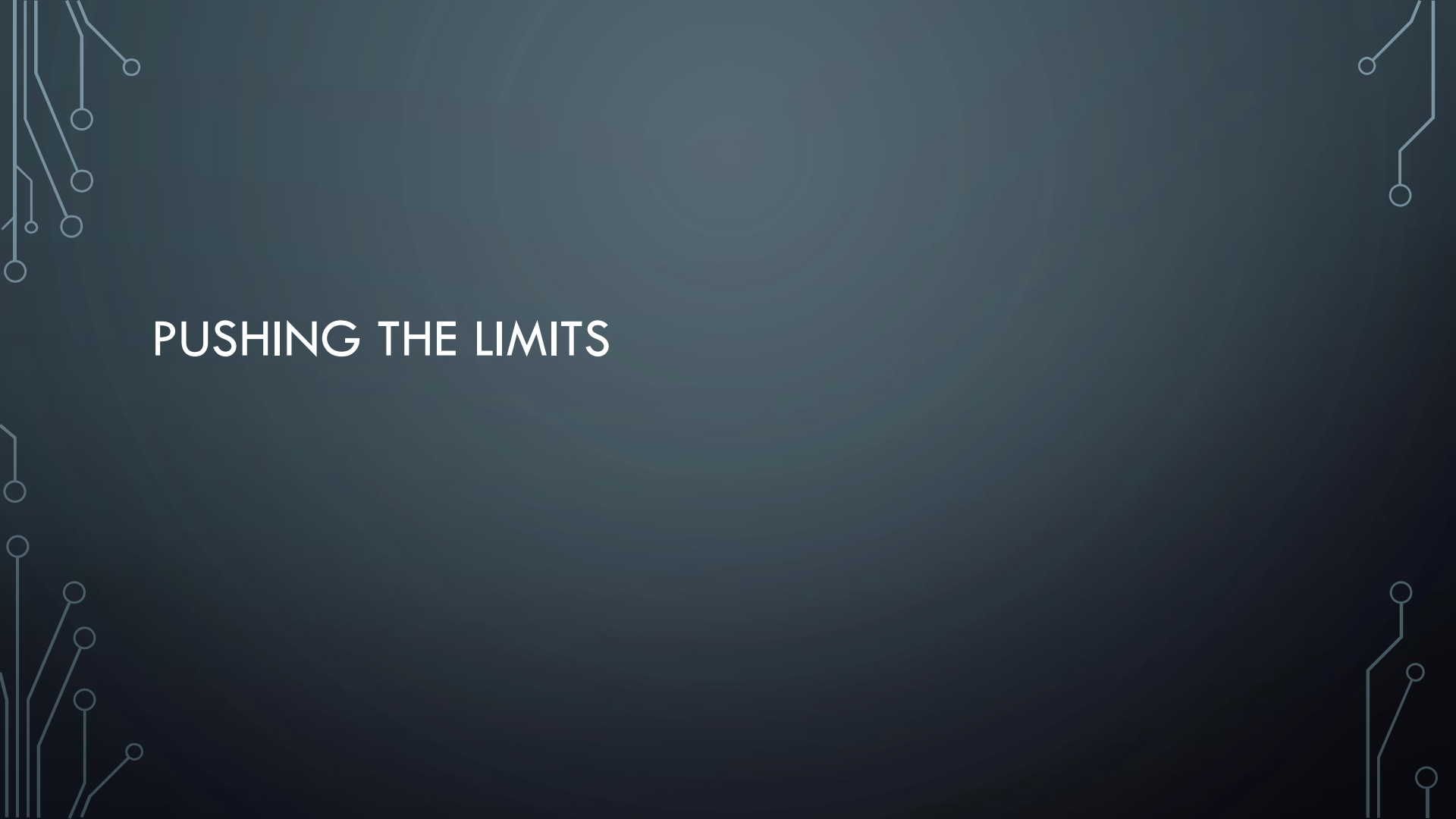
# SUBROUTINES IN “DEMON ATTACK” [UNWIREDBEN]

```
mov r0, [D1Pos]
mov r1, r0
mov r0, [D1Pattern]
mov r2, r0
mov r0, [D1PatIdx]
mov r7, r0
gosub MOVE_DEMON_X
mov r0, r1
mov [D1Pos], r0
mov r0, r7
mov [D1PatIdx], r0
```

Input  
args

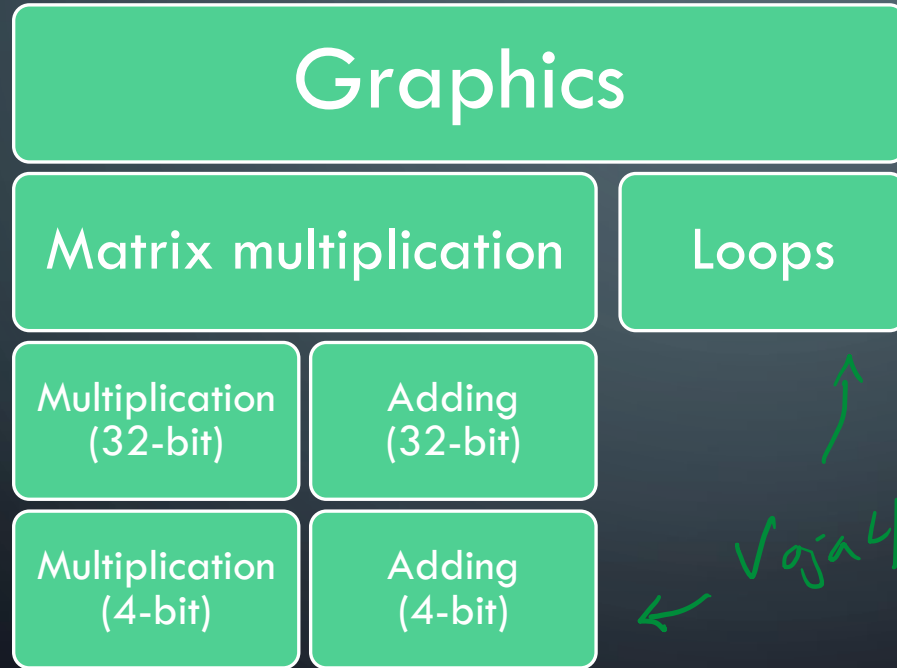
Return  
values





PUSHING THE LIMITS

# THE LIMIT DOES NOT EXIST: TURING-COMPLETENESS



# LINUX ON AN ARDUINO



# HACKADAY

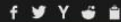
[HOME](#)[BLOG](#)[HACKADAY.IO](#)[TINDIE](#)[HACKADAY PRIZE](#)[SUBMIT](#)[ABOUT](#)

September 9, 2023

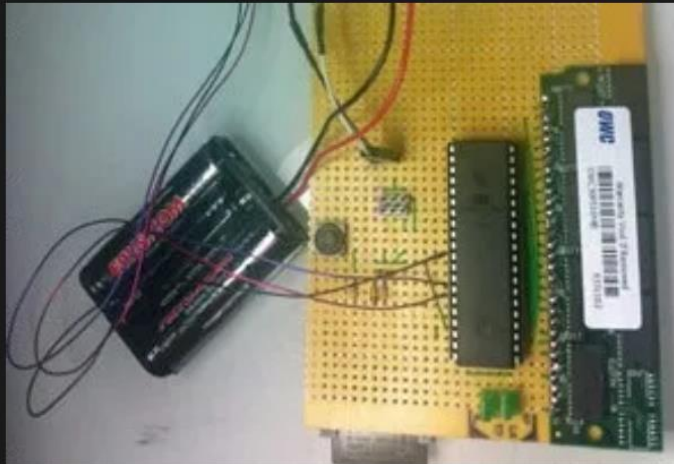
## BUILDING THE WORST LINUX PC EVER

by: [Brian Benchoff](#)

🗨 123 Comments



March 28, 2012



SEARCH

SEARCH

# LINUX ON AN ARDUINO



# HACKADAY

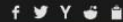
[HOME](#)[BLOG](#)[HACKADAY.IO](#)[TINDIE](#)[HACKADAY PRIZE](#)[SUBMIT](#)[ABOUT](#)

September 9, 2023

## BUILDING THE WORST LINUX PC EVER

by: [Brian Benchoff](#)

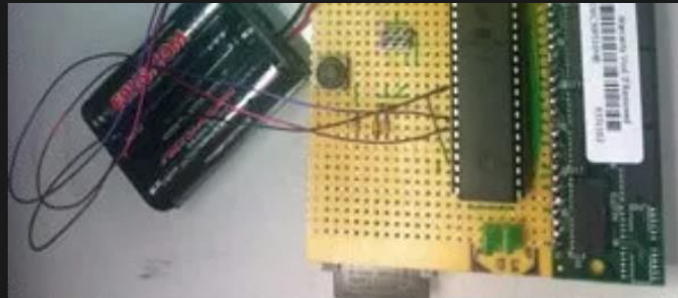
123 Comments



March 28, 2012



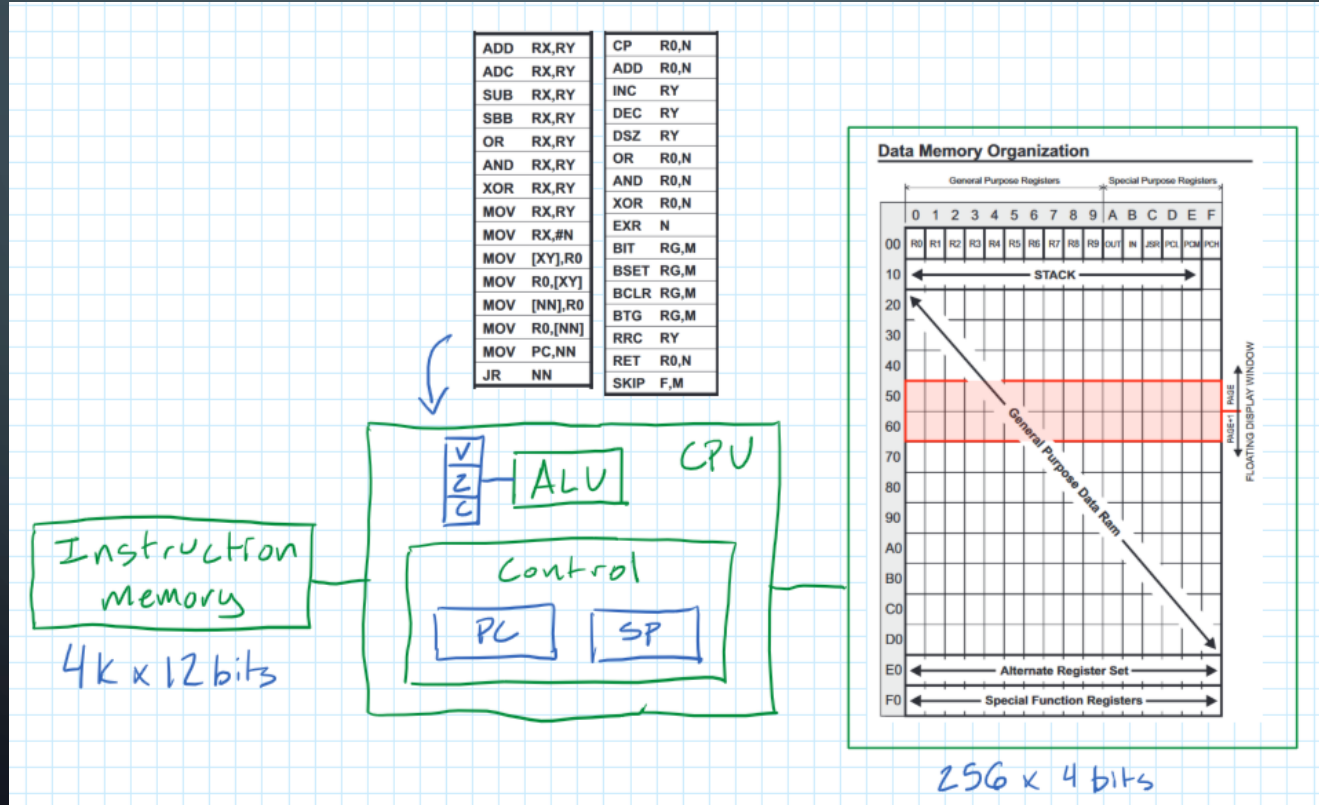
How fast is it? [Dmitry] tells us it takes two hours to boot up to a bash prompt, and four more to load up Ubuntu and login. If you want a Megahertz rating, good luck; the effective clock speed is about 6.5



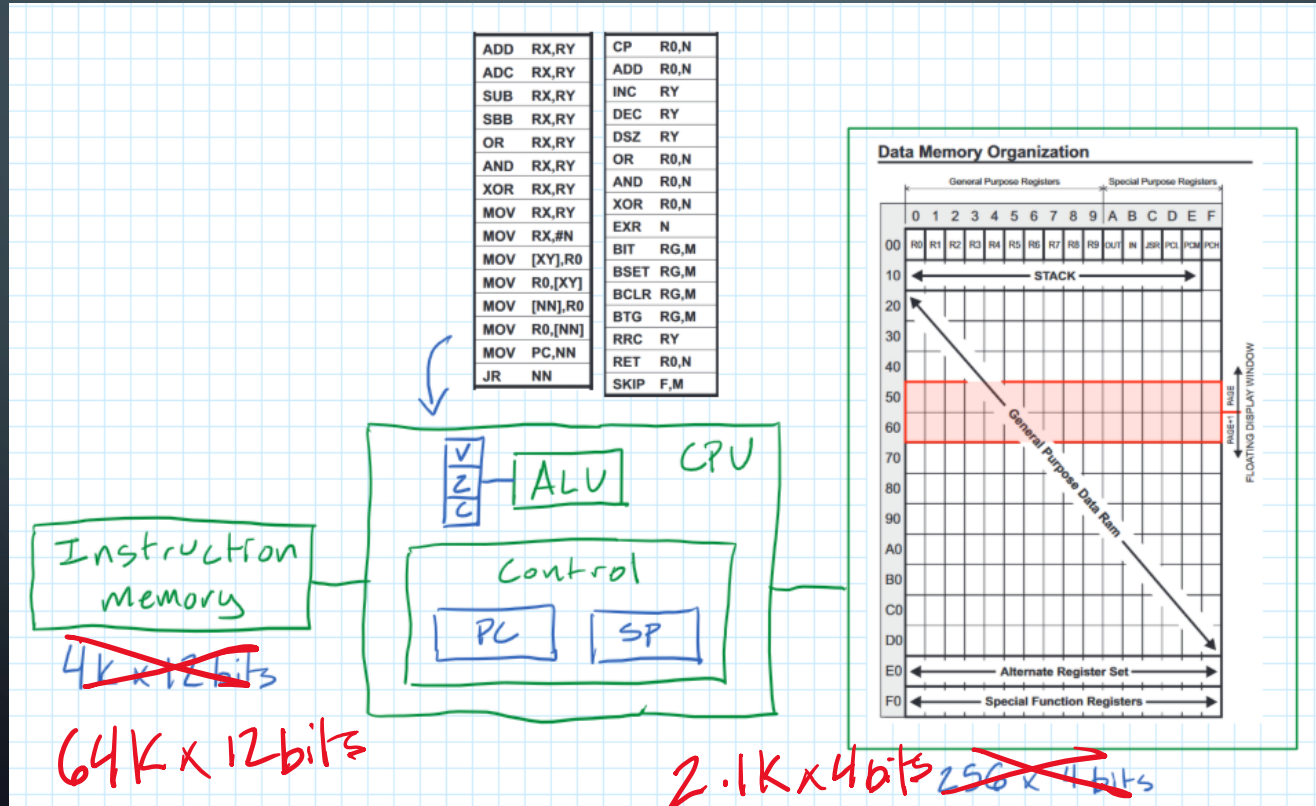
SEARCH

SEARCH

# VOJA4

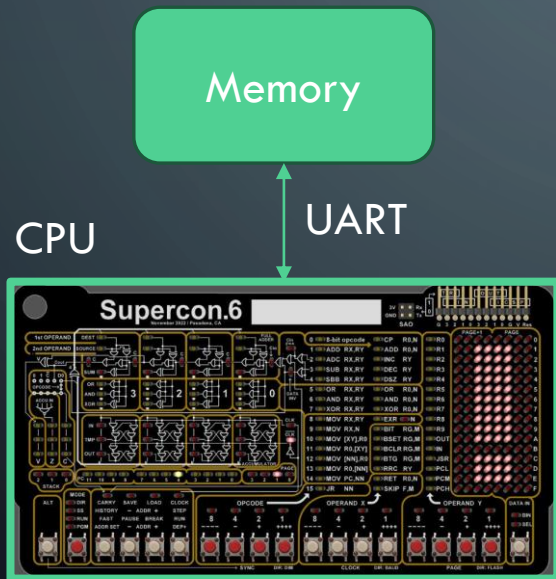


# “VOJA4 XL”



# “VOJA4 XL”

Step 1: Let the Voja4 emulate itself



LOOP:

```
; Read PROGMEM["PC"]  
; "PC" += 1  
; If OP == 0001:  
    ; "RX" += "RY"  
; ...  
; Else if OP == 1010:  
    ; "R0" = DATAMEM["XY"]  
; ...  
JR LOOP
```

# “VOJA4 XL”

Step 1: Let the Voja4 emulate itself

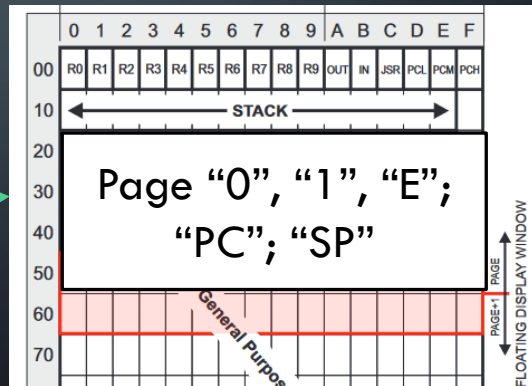
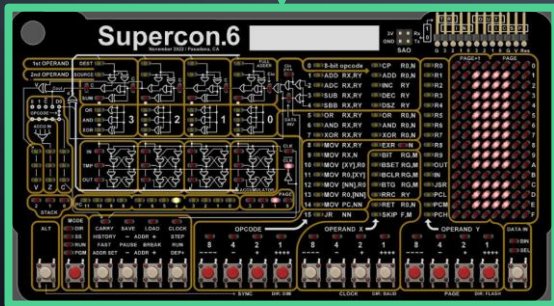
Memory

Instructions  
4k x 12 bits

Data  
192 x 4 bits

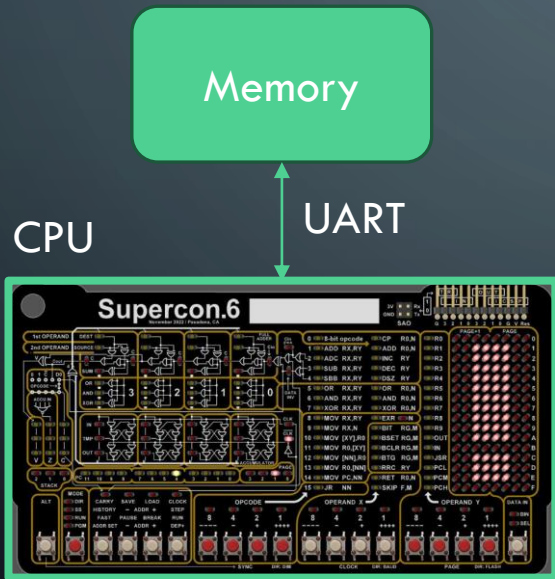
CPU

UART



# “VOJA4 XL”

Step 2: Make PC 16-bits long



16-bits



LOOP:

```
; Read PROGMEM["PC"]  
; "PC" += 1  
; If OP == 0001:  
    ; "RX" += "RY"  
; ...  
; Else if OP == 1010:  
    ; "R0" = DATAMEM["XY"]  
; ...  
JR LOOP
```

# “VOJA4 XL”

Step 2: Make PC 16-bits long

Memory

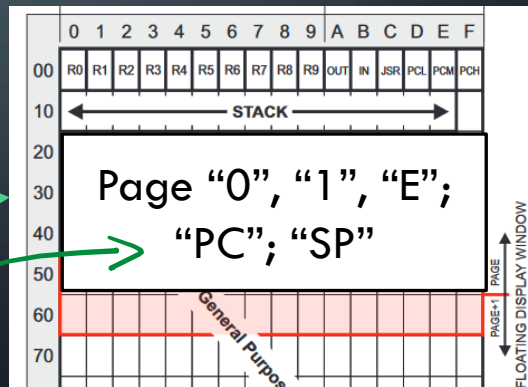
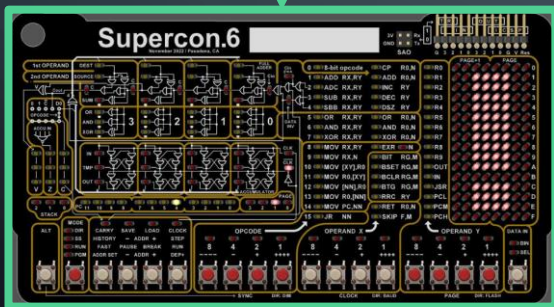
Instructions  
16 blocks x  
4k x 12 bits

Data  
192 x 4 bits

| PC[15:12] | PC[11:0] |
|-----------|----------|
| 0000      | 4k       |
| 0001      | 4k       |
| ...       |          |
| 1111      | 4k       |

CPU

UART



16 bits

# “VOJA4 XL”

Step 3: Extend the ISA

| 8  | 9     | A   | B  | C   | D   | E   | F   |
|----|-------|-----|----|-----|-----|-----|-----|
| R8 | “PCT” | OUT | IN | JSR | PCL | PCM | PCH |

- R9 becomes “PC Top” or “PCT”
- PC is now a 16-bit value formed as [PCT:PCH:PCM:PCL]
- Subroutine calls / long jumps function the same

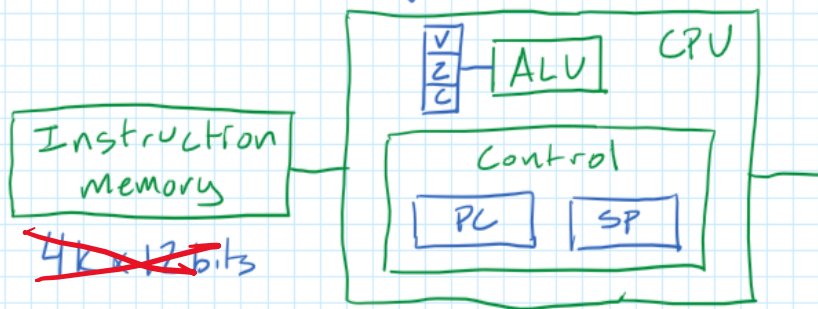
```
MOV R9, Top SQRT
```

```
MOV PC, [High SQRT : Mid SQRT]
```

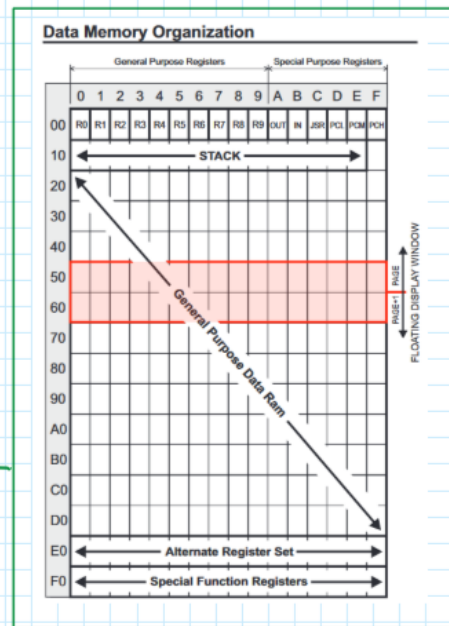
```
MOV JSR, Low SQRT
```

# “VOJA4 XL”

|             |           |
|-------------|-----------|
| ADD RX,RY   | CP R0,N   |
| ADC RX,RY   | ADD R0,N  |
| SUB RX,RY   | INC RY    |
| SBB RX,RY   | DEC RY    |
| OR RX,RY    | DSZ RY    |
| AND RX,RY   | OR R0,N   |
| XOR RX,RY   | AND R0,N  |
| MOV RX,RY   | XOR R0,N  |
| MOV RX,#N   | EXR N     |
| MOV [XY],R0 | BIT RG,M  |
| MOV R0,[XY] | BSET RG,M |
| MOV [NN],R0 | BCLR RG,M |
| MOV R0,[NN] | BTG RG,M  |
| MOV PC,NN   | RRC RY    |
| JR NN       | RET R0,N  |
|             | SKIP F,M  |



~~4K x 12 bits~~  
64K x 12 bits

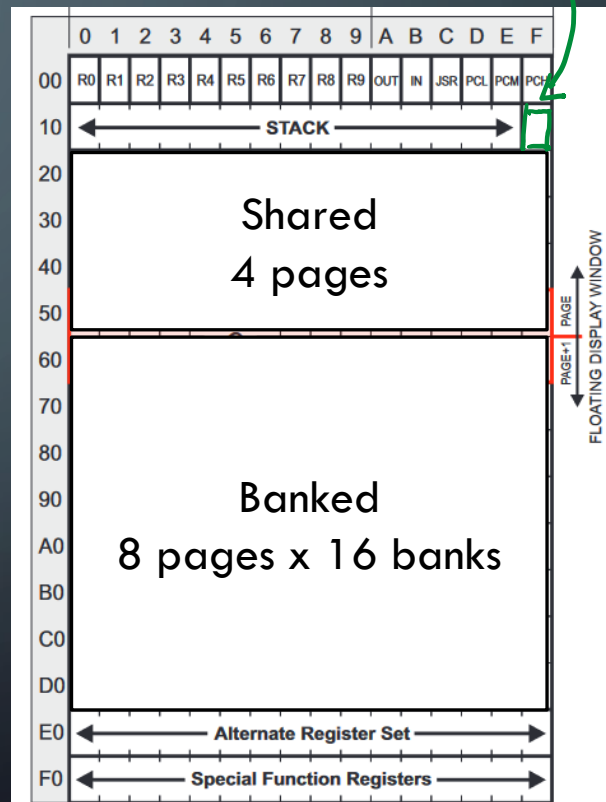
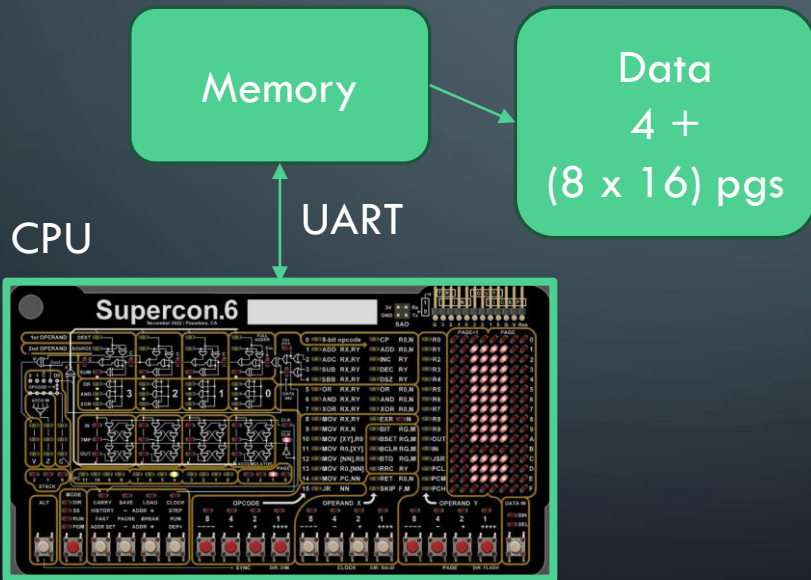


256 x 4 bits

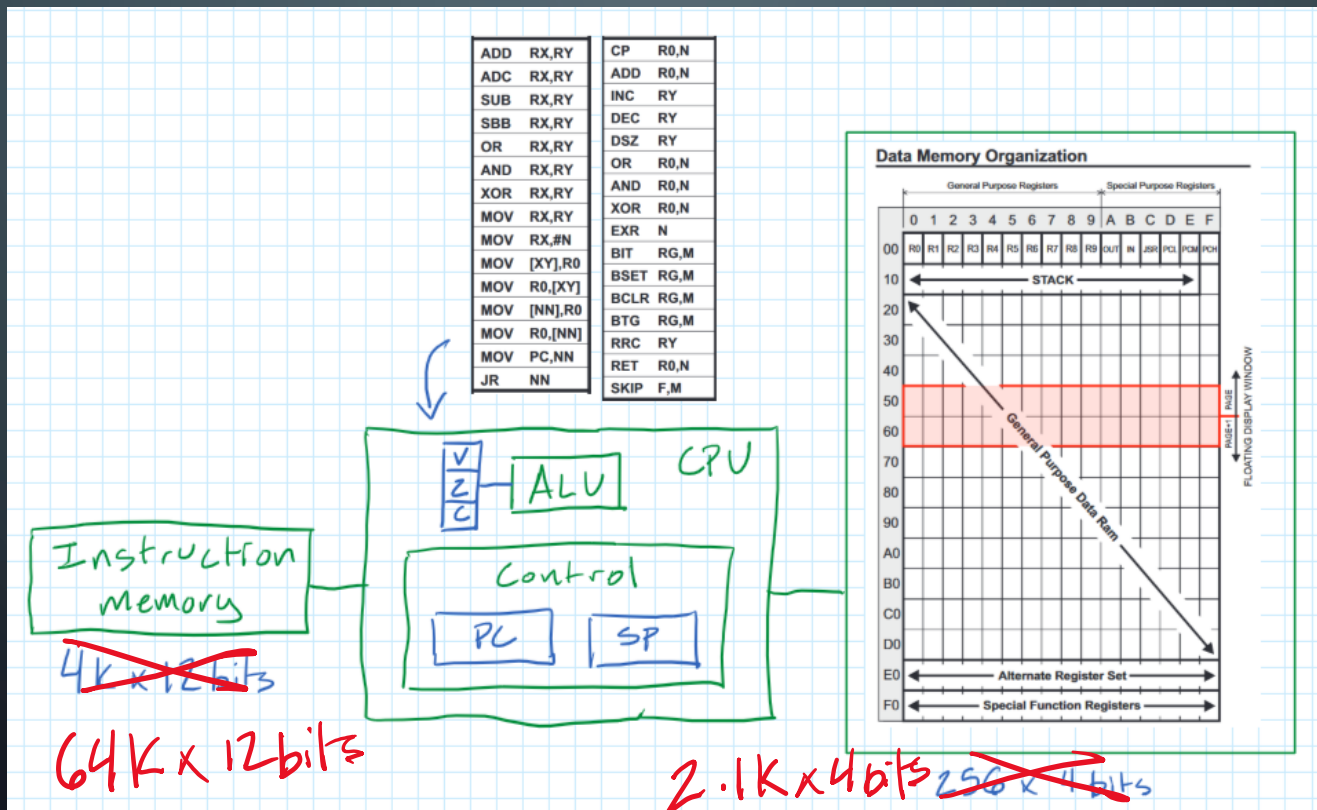
# “VOJA4 XL”

Bank #  
0000

Step 4: Add banked memory



# “VOJA4 XL”



The background is a dark blue gradient. In the corners, there are white line-art illustrations of circuit boards or neural networks, with lines and small circles representing nodes and connections.

## RESOURCES AND GOING FURTHER

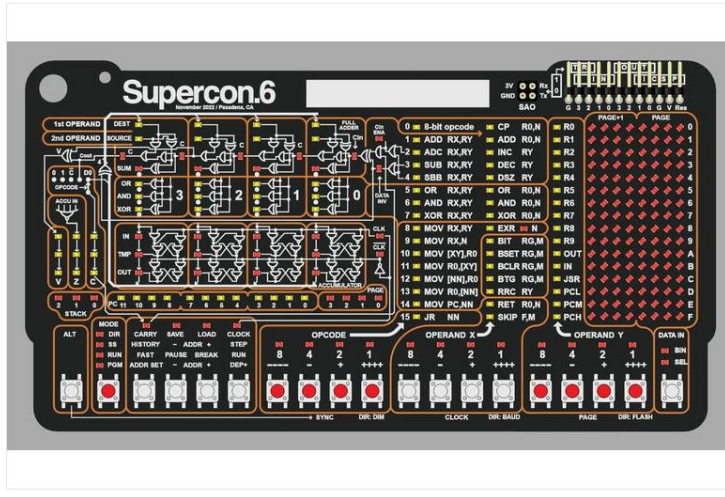
# BUYING THE VOJA4

## Supercon 2022 Badge (Voja4)

Badge from the 2022 Hackaday Supercon

Sold By [0xc0decafe](#)

[Wishlist](#) [Tweet](#) [Share](#) [Pin](#)



\$70.00

[Ask a Question](#)

[No shipping info available.](#)  
[Set destination country to see options](#)

**Out of Stock**

Sign up to get notified when this product is back in stock!

Desired Quantity\*

1

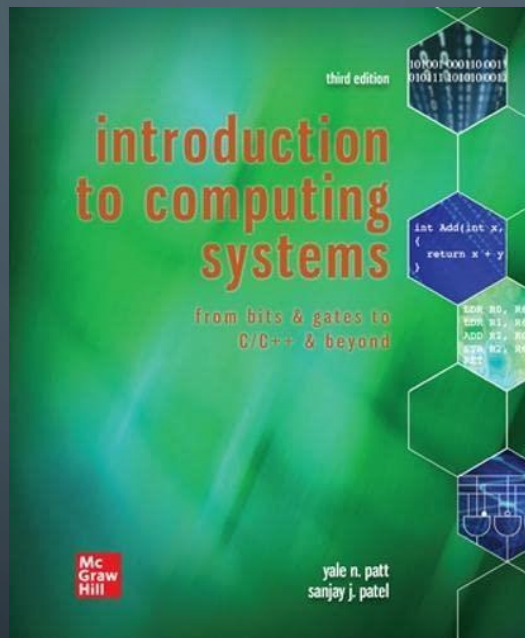
Please message the seller directly if your desired quantity is very large.

<https://www.tindie.com/products/0xc0decafe/supercon-2022-badge-voja4/>

# BOOKS/COURSES

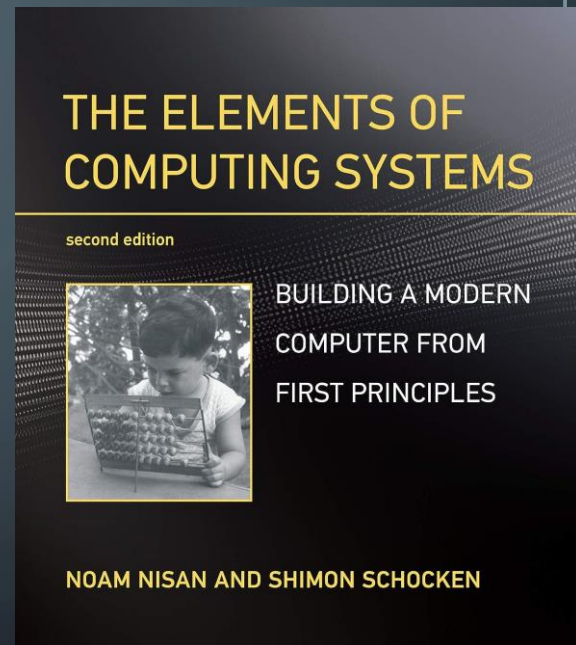


[tinyurl.com/5n6f3zt6](https://tinyurl.com/5n6f3zt6)



[tinyurl.com/39sa23fb](https://tinyurl.com/39sa23fb)

[tinyurl.com/5398wrd5](https://tinyurl.com/5398wrd5)



[tinyurl.com/mrpbnk48](https://tinyurl.com/mrpbnk48)

[nand2tetris.org](https://nand2tetris.org)

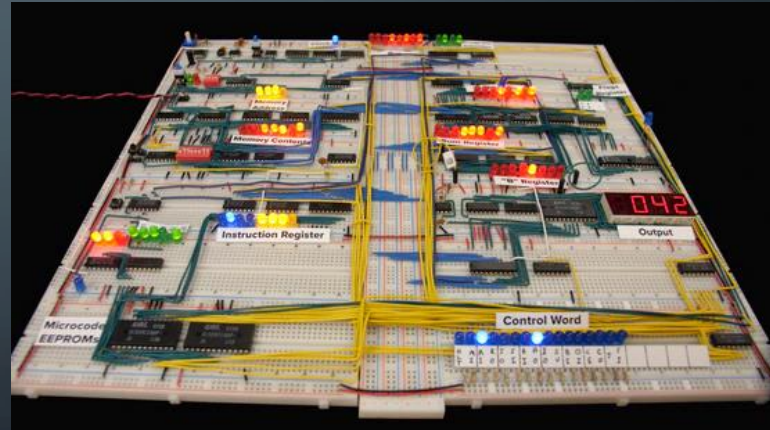
# BREADBOARD COMPUTERS

NIBBLER 4-BIT CPU (FROM BMOW)



[bigmessowires.com/nibbler/](http://bigmessowires.com/nibbler/)

BEN EATER'S 8-BIT CPU



[eater.net/8bit/kits](http://eater.net/8bit/kits)

THANKS!

Voja Antonic



**Supplyframe**