

## POPCORN V3 SYNTHESIZABLE 8-BIT CISC MICROPROCESSOR

JEUNG JOON LEE  
5/2000

*"possibly world's simplest, smallest yet capable 8 bit CISC."*

### 1.0 POPCORN V3 FEATURES

- Architecture and instruction set is a cross between Microchip's PIC and Atmel's AVR processors.
- Fully self-contained 8 bit CISC microprocessor IP, synthesizable on most medium capacity CPLD or small capacity FPGA. *Possibly world's simplest, smallest yet capable 8 bit CISC.*
- Accumulator based architecture keeps software complexity low while minimizing synthesizable gate count.
- Flexible external stack implementation with internal scalable stack pointer.
- Easy to learn instruction set, similar to that of Intel 8085, featuring 43 instructions with 5 addressing modes.
- Opcode length of one to three bytes
- Opcode execution cycle of 5 to 17.
- Scalable memory space from 256 bytes to 64Kbytes.
- Scalable port memory space from 256 bytes to 64Kbytes.
- Scalable internal register file
- Full wait-state capable. Slow memories such as DRAM controller, can interface PopCorn-V3 using the READY\_H pin.
- Streamline single clock architecture ( PopCorn v1 used two out of phase clocks ) allows higher system clock rate.
- Synchronous maskable interrupt support, with programmable interrupt vector. The interrupt can also be generated internally (via software).
- Built-in boot capability, which allows copying of code in FLASH/EEPROM to SRAM for faster execution. Amount of copying is scalable.

### 2.0 SYSTEM OVERVIEW

PopCorn is a complete 8 bit CISC microprocessor IP core. With the addition of up to 64Kbyte of code RAM and up to 64Kbyte of SRAM, a complete computer system can be implemented.

PopCorn's architecture and instruction-set has been *purposefully* minimized to reduce gates count, making it ideal to target low-cost FPGA/CPLDs.

Lattice Semiconductor ispLSI 3256 CPLD (11K gate)  
Xilinx Semiconductor XC3030 FPGA (3K gate)  
Altera Semiconductor EPF10K10 FPGA

Due to the modular design, multiple lower capacity devices can be used. Three modules comprises the PC core:

#### PC.v

This is the highest level module. It serves as a wrapper of the other two modules, as well as providing two clock sources to the modules.

#### DATAPATH.v

This module is the datapath of the processor. It contains all of the onboard registers, ALU, program and stack pointers.

#### SEQUENCER.v

This module contains the sequencing state machine responsible for the proper decode and execution of the opcodes.

#### BUSCON.v

This module contains the address and control interface to the external world.

#### INC.h

Contains parameterizable data as well as feature enables.

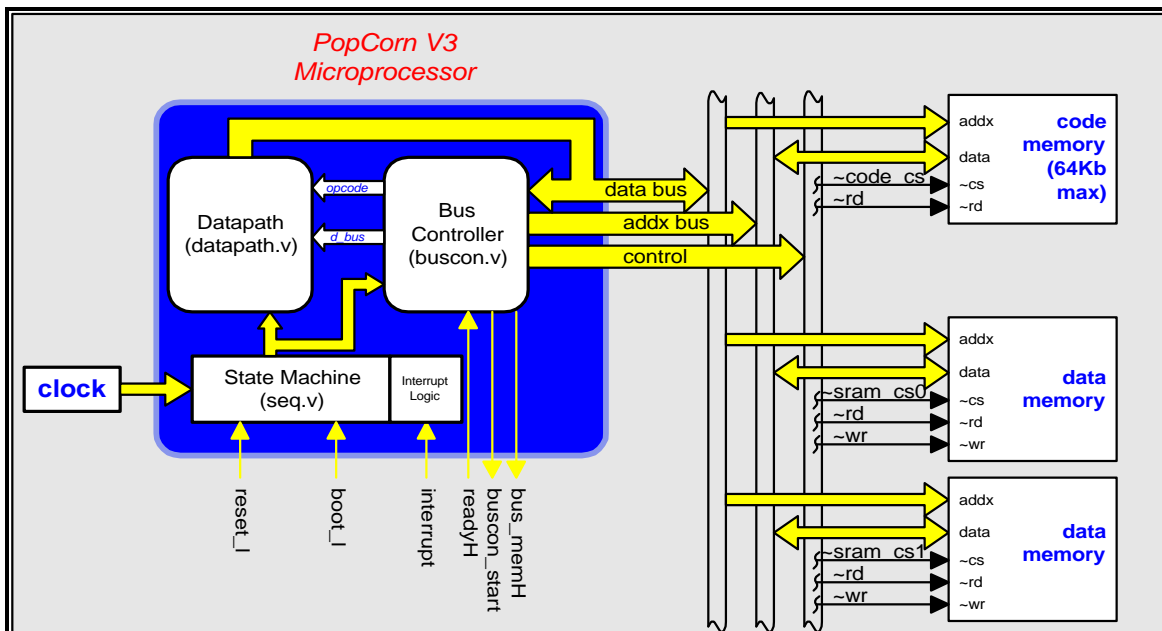


Figure 1 This diagram illustrates a typical 8 bit computer based on PopCorn V3 8-bit CISC processor. Notice the 3 main blocks comprising PopCorn V3 processor.

### 3.0 IP PINOUT DESCRIPTION

Pin	Dir	Description
sys_rst_l	I	Main system reset, asynchronous active low. When reset is asserted all internal registers are cleared (0) and all state machines are placed in their initial state, and interrupts are disabled.
sys_clk	I	Main clock source. All state machines advances on the rising edge of this clock.
addx_bus [ `AW-1:0 ]	O	This is the address bus driven by PopCorn-V3 during a bus cycle. It is stable over the entire bus cycle. This bus is scalable from 8 to 16 bits. See text.
data_bus[7:0]	I/O	This is the bidirectional data bus to the core. Through the use of 'inc.h' file it is possible to use unidirectional data bus.
code_cs_L	O	This is an active low chip-select asserted when lower ½ of address space is accessed without the "booting option". This address space is assumed to be "code" space. See text.
sram_cs0_L	O	This is an active low chip-select asserted when lower ½ of address space is selected when "booting" option has been selected. See text.
sram_cs1_L	O	This is an active low chip-select asserted when the upper ½ of address space is selected. This address space is assumed to be "data" space. See text
wr_L	O	Active low signal which indicates that the current bus cycle is write. All slave devices, including memory, should use the rising edge of this signal to capture the data drive in the data_bus[7:0]
rd_L	O	Active low signal which indicates that the current bus cycle is read. PopCorn uses the rising edge of this signal to capture data being presented by slave devices on data_bus[7:0].
ready_H	I	Active high input signal which indicates that external slave devices are "ready" and thus do not need the bus to insert wait states. When PopCorn samples this signal as low, wait-states will be inserted until it becomes high.
int_L	I	Active low input signal which when sampled low on the rising edge of sys_clk, indicates interrupt to PopCorn. It is only necessary to assert it for 1 sys_clk. Logic internal to PopCorn will hold sampled until recognized by the processor.
boot_l	I	Active low input which indicates that "boot copying" is to be performed following a system reset. See text.
bus_memH	O	Output which when high indicates that the present bus cycle is "memory", as opposed to "port".

### 4.0 BUS CYCLES

PopCorn-V3 processor has a simple bus cycle, as shown in figures 2 and 3. Wait state insertion is also straightforward.

A typical read cycle, with no wait state, consists of 3 *sys\_clk* cycles, as shown in figure 2. There are 5 critical parameters, *tas*, *tcs*, *trs*, *tds* and *tdh*. Most of these parameters will depend on the synthesis, and constraint settings. In order to minimize gate count, most of the bus signal being drive by PopCorn-v3 are not flopped, that is they *may come directly from a combinatorial output*. Note that this is not a design limitation, rather a conscious choice in design performance & resource tradeoff. With a reasonable clock frequency, this may not impact the interface with external devices.

During the T1 cycle, stable address and chip select signals are output to the bus. During the Tw cycle, *rd\_l* or *wr\_l* signal is asserted. On the T2 cycle, *rd\_l* or *wr\_l* is deasserted.

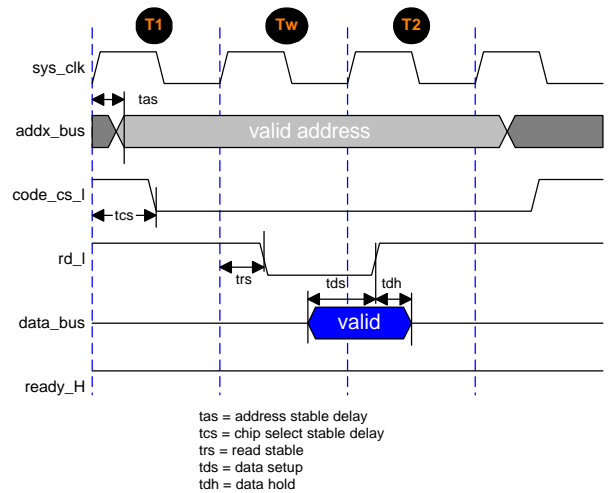


Figure 2 Typical read cycle consists of 3 cycles. Notice that there is no wait cycle.

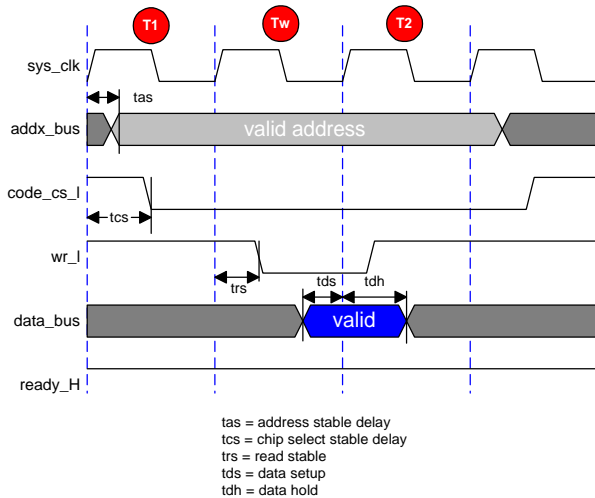


Figure 3 Typical write cycles consists of 3 cycles. There is no wait state as shown.

When wait state is needed, the external device needs to pull low `ready_H` as soon as the address is decoded. PopCorn-V3's bus controller samples the state of `ready_H` starting on the second rising edge after stable address. If it is logic low, then a wait state is inserted. Wait states will be inserted for as long as `ready_H` is sampled low. In the example of figure 4, 2 wait states are inserted, resulting in overall bus cycle of 5 cycles.

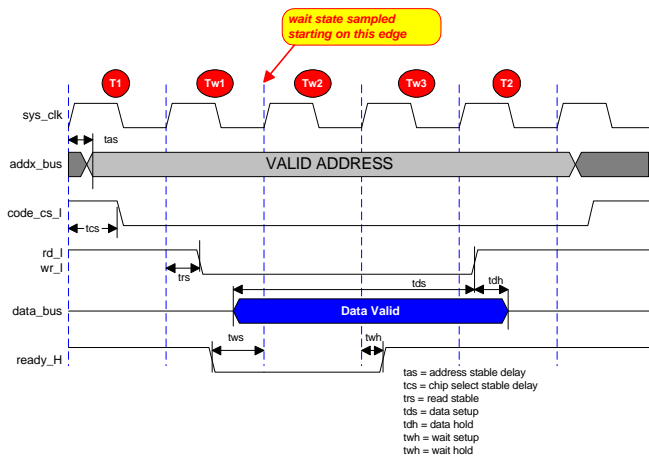


Figure 4 If wait state is needed, the external device needs to pull low on `ready_H` as soon as a valid address is asserted.

## 5.0 INTERRUPT PROCESSING

PopCorn-V3 supports a flexible interrupt logic to allow processing real-time events with minimum latency. Interrupts can be generated by the software, or by external devices through the `int_L` pin.

The default core contains only one interrupt input pin, but through the use of peripherals (interrupt controller) multiple interrupt sources can be accommodated.

The Interrupt Vector is hard-coded to address, 0x0010. The content of this address is where the processor core jumps to after context switch.

### 5.1 Hardware Interface

PopCorn-V3 samples the state of `int_L` on every rising edge of `sys_clk`. When a low logic level is sampled, the "interrupt status holding flop" is set, see figure 5. Because PopCorn-V3 samples and queues the interrupt request, `int_L` needs to be low for just 1 `sys_clk` cycle. A valid "interrupt request" signal will be sent to the processor core if "interrupt mask flop" is also set.

When the processor core services the "interrupt status", the "interrupt mask flop" is automatically reset. That is, the hardware automatically disables further interrupts. Thus if interrupts are to be desired beyond this point, the firmware must execute EI instruction to set the "interrupt mask flop".

The "interrupt status holding flop" is not reset by the hardware, so it is the responsibility of the ISR (interrupt service routine) to clear this flop before exiting so as to prepare for the next interrupt event. Failure to do so can create multiple back-to-back interrupts.

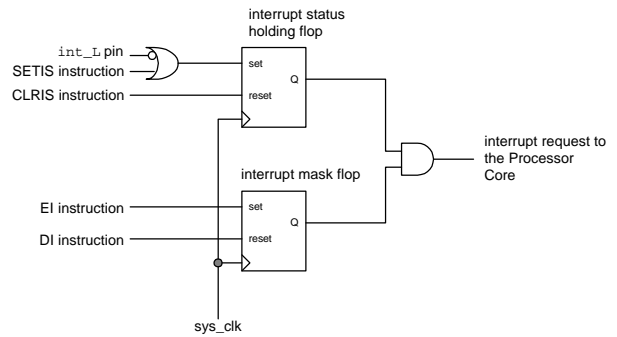


Figure 5 Structure of PopCorn-V3 interrupt handling

### 5.2 Software Interface

There are 4 instructions associated with the interrupt:

**EI** - Enables global interrupt. This instruction sets the "interrupt mask flop", upon the completion of the instruction following EI. This is designed such that an interrupt can not occur while returning from an ISR (interrupt service routine) through the RET instruction.

**DI** - Disables the global interrupt. This instruction clears the "interrupt mask flop". Once this flops is cleared, no interrupt request can occur to the processor core.

**SETIS** - Set Interrupt Status. This instruction, when executed, will set the "interrupt status holding flop". It has the same effect as if the `int_L` pin was sampled low. Firmware can use this instruction to generate a software interrupt.

**CLRIS** - Clear Interrupt Status. This instruction clears the "interrupt status holding flop". Note that once the "interrupt status holding flop" has been set, the only way to reset is to execute this instruction. Failure to do so can generate multiple back-to-back interrupts to the processor core.

The below is an assembly core fragment showing a typical and proper interrupt code style.

Typical ISR code

```

; ISR Code begins here
ISR_Code:
    PUSH flag ;save flag register

    ; Body of ISR starts here
    PUSH acc ;save acc if needs to be
    .
    .
    POP acc ; restore acc
    ; Body of ISR ends here

CLRIS ; reset interrupt status
      ; holding register
EI ; re-enable global
   ; interrupt, after RET
RET ; return from ISR.
   ; Interrupt mask flop
   ; is not set until the
   ; execution of RET
   ; instruction
    
```

Main Code:

```

EI ;enable global interrupt
xx: JMP xx ;wait for an interrupt
   ;from pin
    
```

The worse case interrupt latency is computed as follows:

$$t_{IL} = \text{Longest Instruction} + 4 \text{ cycles}$$

Interrupt latency is defined as the time from the request of interrupt by an external device to the execution of first instruction in ISR.

**5.3 Summary of Interrupt Event**

The sequence of events which takes place when an interrupt is serviced is summarized below:

1. "Interrupt Request" is sampled by the processor core on every EXECUTE state (see figure 7, sequencer state machine)
2. A bus cycle is issued to fetch the low byte Interrupt Vector from memory address 0x0010
3. A bus cycle is issued to fetch the high byte Interrupt Vector from memory address 0x0011.
4. The processor core resets the "interrupt mask flop" to prevent further interrupts.
5. Push PC (program counter) high byte to the stack
6. Push PC low byte to the stack
7. Jump to Interrupt Vector, fetched from address 0x0010 and 0x0011.

**6.0 BOOT UP**

PopCorn-V3 contains a boot-up feature designed specifically to increase code execution speed. In a nutshell, the Boot-Up process consists of "code shadowing". That is, the firmware residing in a EEPROM or FLASH is copied into a SRAM mapped to the same address. Once this copying process is finished code execution begins from the SRAM. The boot-up process relies on the fact that SRAM access time is by far smaller than the access time of EEPROM or FLASH. Typical "fast" SRAM access time is in the

order of 15 to 25nS, while typical EEPROM/FLASH access time is 100 to 200nS. This allows the processor to be clocked at high frequency without the need to insert wait states for code access.

Upon system reset, the processor core samples the logic state of boot\_1 pin. If this pin is active low, copying process commences from FLASH/EEPROM to SRAM. The processor core inserts 7 wait states every time FLASH/EEPROM is accessed. This guarantees that even the most slow devices can be used in the boot-up process.

The boot-up configuration is glueless due to the provision of dedicated chip-select output pins. Refer to figure 6. A typical configuration consists of 3 memory devices: SRAM0, SRAM1 and 1 EEPROM/FLASH. The SRAM0 and EEPROM/FLASH shares the same lower 32Kbyte of address space. The SRAM1 occupies the upper 32Kbyte of the address space. The booting process consists of copying the content of the FLASH/EEPROM to SRAM0. Once the copying process is finished, code execution begins from SRAM0. Only sram\_cs0\_1 and sram\_cs1\_1 are used from this point forward.

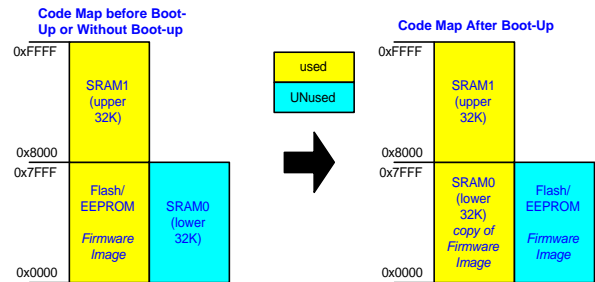
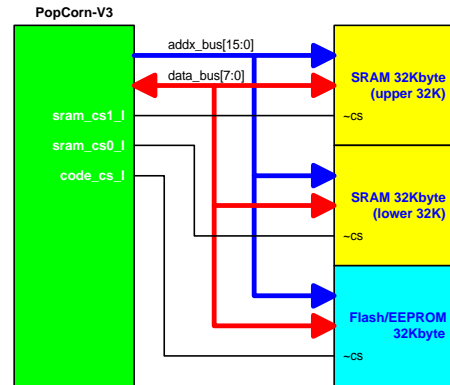


Figure 6 Typical configuration for Boot-Up. See text

**7.0 CORE FUNCTIONAL DESCRIPTION**

PopCorn-V3 is composed of 3 logically partitioned block. Each block is composed of one Verilog module.

**7.1 Sequencer - Seq.v**

The sequencer is the main control state machine. Figure 8 illustrates the state flow diagram. The state machine is responsible for controlling the datapath and the bus controller in order to fetch an instruction, decode it and execute it.

7 states comprises the state machine. The reset default state is "state\_execute".

#### state\_execute

In this state, the previously loaded and decoded opcode is executed. Opcode execution refers to updating of internal registers, such as flags and accumulators. The next state is always "state\_opcode\_operand" in which the next opcode is fetched.

#### state\_opcode\_operand

This state serves two purposes. It issues a command to the bus-controller to load from memory an opcode or an operand

state\_decode

state\_load\_hi

state\_read\_mem\_lo

state\_read\_mem\_hi

state\_write\_mem\_lo

state\_write\_mem\_hi

## 7.2 Datapath - Datapth.v

Datapath contains the data flow logic. Refer to figure 7. An ALU, and the register file comprises the heart of datapath. The register file is reduced to the bare minimum. There are: accumulator or ACC, AX register, BX register, PORT register and FLAG register.

The ACC is treated specially in that many instructions can perform operations with it only. These includes most of the arithmetic instructions.

AX, BX and PORT registers are general purpose registers, and the firmware can use them for any purpose.

The FLAG register is 3 bits and holds the ALU result of the last executed instruction. The FLAG register can be written by the firmware.

## 7.3 Bus Controller - Buscon.v

### **internal registers**

#### **the command meaning (the bits)**

the bus controller is responsible for managing the bus cycle to the external world. This includes in the housekeeping of the Program Counter (PC), the Stack Pointer (SP), the opcode holding register (OPL), and the operand holding registers (OPHI , OPLO). A state machine

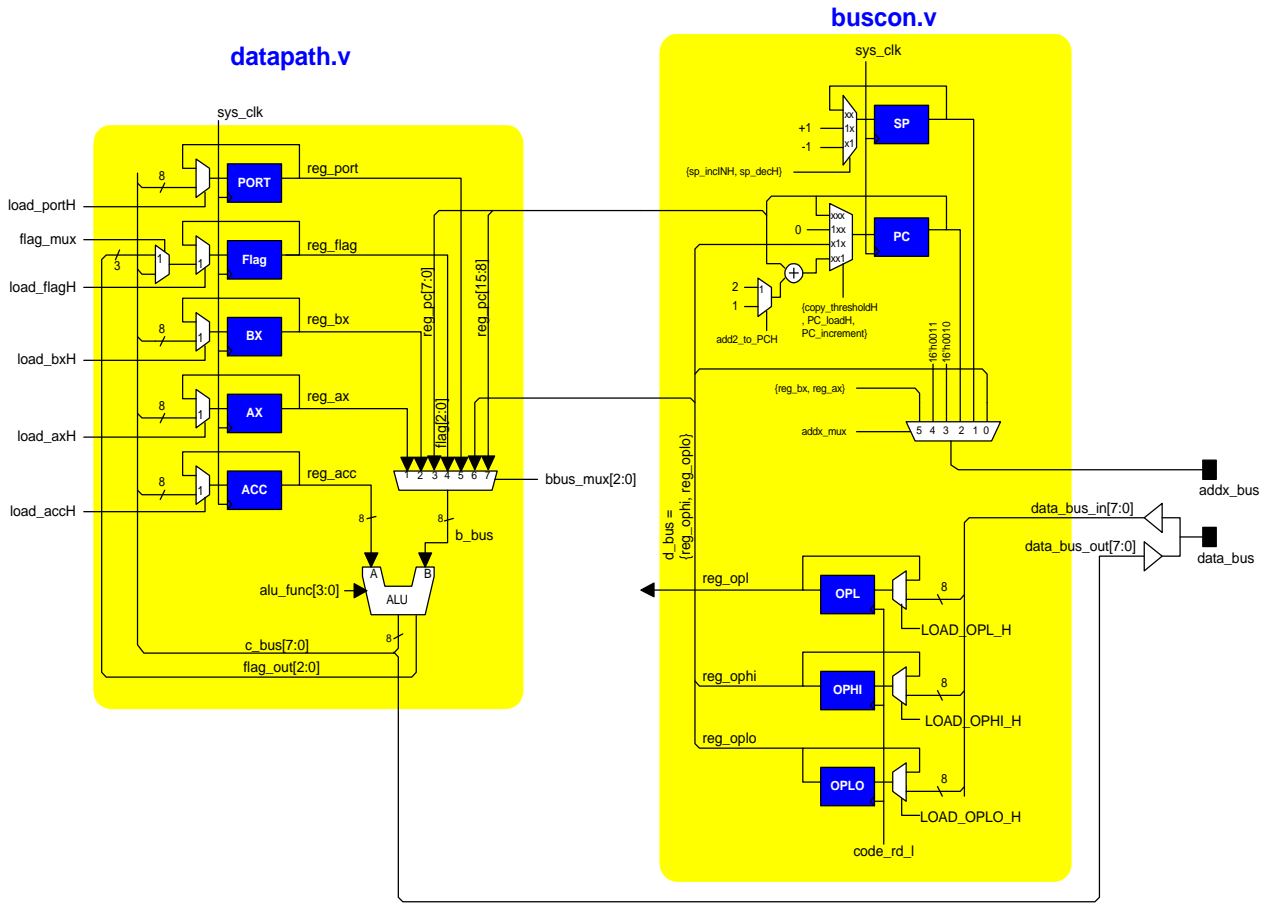


Figure 7 Popcorn.v Datapath block diagram

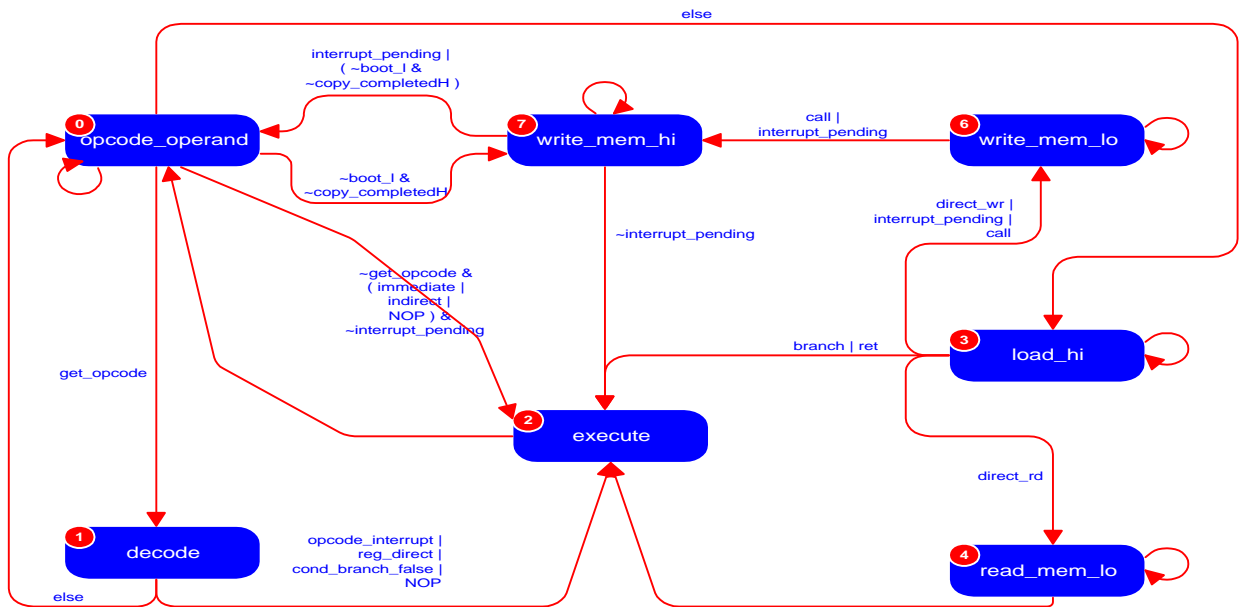


Figure 8 Sequencer state machine

**software overview:**  
**data types**

**addressing modes**

PopCorn V3 supports 5 types of addressing modes:

Immediate

Register Direct

Direct

Indirect

Indexed

**flag**

**instruction set**

**Assembler:**

- syntax
- how it works
- restrictions
- chars/line, etc

Nomenclature	Description
AX	General Purpose register AX
BX	General Purpose register BX
FLAG	Status register
ACC	Accumulator
r	Destination/Source register which can be: AX,BX, FLAG or PX
c	carry flag bit
d	Destination/Source register which is the same as "R", but includes ACC
temp	temporary
#8	8 bit immediate data
#16	16 bit immediate data

value	r r r	d d d
0 0 0	-	ACC
0 0 1	AX	AX
0 1 0	BX	BX
0 1 1	-	-
1 0 0	FLAGS	FLAGS
1 0 1	PORT (P)	PORT (P)
1 1 0	-	-

1 1 1 - -

---



MNEMONIC	BYTES	CYCLES	MODE	OPERATION	OPCODE	FLAGS AFFECTED
<b>Data Transfer</b>						
LDI d,#8	2	8	immediate	loads #8 into d register	1011 1ddd	
LDACC r	1	5	reg direct	copies r to ACC	1000 1rrr	
STACC r	1	5	reg direct	copies ACC to r	1001 0rrr	
LDM d, #16	3	11	direct	loads d with data present at memory location #16	0110 0ddd	
STM d, #16	3	11	direct	stores d to memory location at #16	0110 1ddd	
LDP d, #16	3	11	direct	loads d with data at port address at #16	0101 0ddd	
STP d, #16	3	11	direct	stores d to port address at #16	0101 1ddd	
LDACCAB	1	8	indexed	loads ACC with data present at memory location pointed by content of register pair BX, AX	0111 1110	
STACCAB	1	8	indexed	stores ACC to memory location pointed by content of register pair BX, AX	0111 0110	
<b>Arithmetic Operations</b>						
ADDACCI #8	2	8	immediate	ACC = ACC + #8	0000 0111	Z,P,CY
ADDACCIC #8	2	8	immediate	ACC = ACC + #8 + c	0100 0111	Z,P,CY
SUBACCI #8	2	8	immediate	ACC = ACC - #8	0000 1111	Z,P,CY
ANDACCI #8	2	8	immediate	ACC = ACC & #8	0001 0111	Z,P,CY
ORACCI #8	2	8	immediate	ACC = ACC   #8	0001 1111	Z,P,CY
XORACCI #8	2	8	immediate	ACC = ACC ^ #8	0010 0111	Z,P,CY
CMPACCI #8	2	8	immediate	temp = ACC - #8	0100 1111	Z,P,CY
ADDACC r	1	5	reg direct	ACC = ACC + r	0000 0rrr	Z,P,CY
ADDACCC r	1	5	reg direct	ACC = ACC + r + c	0100 0rrr	Z,P,CY
SUBACC r	1	5	reg direct	ACC = ACC - r	0000 1rrr	Z,P,CY
ANDACC r	1	5	reg direct	ACC = ACC & r	0001 0rrr	Z,P,CY
ORACC r	1	5	reg direct	ACC = ACC   r	0001 1rrr	Z,P,CY
XORACC r	1	5	reg direct	ACC = ACC ^ r	0010 0rrr	Z,P,CY
NOTACC r	1	5	reg direct	ACC = ! r	0010 1101	Z,P,CY
SHRACC r	1	5	reg direct	ACC = r >> 1; CY = ACC[0]; ACC[7] = 0	0011 0101	Z,P,CY
SHLACC r	1	5	reg direct	ACC = r << 1; CY = ACC[7]; ACC[0] = 0	0011 1101	Z,P,CY
RORACC r	1	5	reg direct	ACC = r >> 1; ACC[7] = ACC[0]	0011 0110	Z,P,CY
ROLACC r	1	5	reg direct	ACC = r << 1; ACC[0] = ACC[7]	0011 1110	Z,P,CY
CMPACC r	1	5	reg direct	temp = ACC - r	0100 1rrr	Z,P,CY
<b>Branching</b>						
JE #n16	3	11/5	direct	PC = #16 if Z=1	1001 1011	
JNE #16	3	11/5	direct	PC = #16 if Z=0	1010 0011	
JP #16	3	11/5	direct	PC = #16 if P=1	1010 1011	
JN #16	3	11/5	direct	PC = #16 if P=0	1011 0011	
JC #16	3	11/5	direct	PC = #16 if CY=1	1011 1011	
JNC #16	3	11/5	direct	PC = #16 if CY=0	1100 0011	
JMP #16	3	11	direct	PC = #16	1100 1011	
CALL #16	3	17	direct	PC = #16; [SP] = PC; SP=SP+2	1101 0110	
RET	3	11	direct	SP = SP - 2; PC = [SP]	1101 1110	
PUSH d	1	8	indirect	[SP] = d; SP = SP + 2	0111 0ddd	
POP d	1	8	indirect	SP = SP + 2; d = [SP]	0111 1ddd	
<b>Special</b>						
EI		5		Enable global interrupt	1110 0111	
DI		5		Disable global interrupt	1110 1111	
NOP		5		Does nothing	0000 0000	
SETIS		5		Sets the interrupt status. If global interrupt is enabled, this will cause an interrupt.	1111 0111	
CLRIS		5		Clears the interrupt status.	1111 1111	

Detail Opcode Description

---

**ADDACCI #8**

cycles: 8 cycles  
bytes: 2 bytes = 1 opcode + 1 8bit operand  
operation: ACC = ACC + #8

This instruction adds the current value of the accumulator with an 8 bit immediate value and places the result in to the accumulator.

opcode:

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

n	n	n	n	n	n	n	n
---	---	---	---	---	---	---	---

examples:

```
ldi    acc,#20 ; acc = 20h
addacci #20h ; acc = 40h
```

---

**ADDACCIC #8**

cycles: 8 cycles  
bytes: 2 bytes = 1 opcode + 1 8bit operand  
operation: ACC = ACC + #8 + CY

This instruction adds the current value of the accumulator with an 8 bit immediate value plus the current state of carry flag bit, CY. The result is placed back to the accumulator.

opcode:

0	1	0	0	0	1	1	1
---	---	---	---	---	---	---	---

n	n	n	n	n	n	n	n
---	---	---	---	---	---	---	---

examples:

```
ldi    flag,#04 ; CY = 1
addaccic #20h ; acc = 21h
```

---

**SUBACCI #8**

cycles: 8 cycles  
bytes: 2 bytes = 1 opcode + 1 8bit operand  
operation: ACC = ACC - #8

This instruction subtracts the immediate 8 bit value from the current content of the accumulator. The result is placed back to the accumulator.

opcode:

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

n	n	n	n	n	n	n	n
---	---	---	---	---	---	---	---

examples:

```
ldi    acc,#20h ; acc = 20h
subacci #21h ; acc = FFh
```

---

**ANDACCI #8**

cycles: 8 cycles  
bytes: 2 bytes = 1 opcode + 1 8bit operand  
operation: ACC = ACC & #8

This instruction performs a logical bit wise AND operation of the current content of the accumulator with an 8 bit immediate value. The result is placed back to the accumulator.

opcode:

0	0	0	1	0	1	1	1
---	---	---	---	---	---	---	---

n	n	n	n	n	n	n	n
---	---	---	---	---	---	---	---

examples:

```
ldi    acc,#F0h ; acc = F0h
andacci #20h ; acc = 20h
```

---

**ORACCI #8**

cycles: 8 cycles  
bytes: 2 bytes = 1 opcode + 1 8bit operand  
operation: ACC = ACC | #8

This instruction performs a logical bit wise OR operation of the current content of the accumulator with an 8 bit immediate value. The result is placed back to the accumulator.

opcode:

0	0	0	1	1	1	1	1
---	---	---	---	---	---	---	---

n	n	n	n	n	n	n	n
---	---	---	---	---	---	---	---

examples:

```
ldi    acc,#06h ; acc = 06h
oracci #F0h ; acc = F6h
```

---

**XORACCI #8**

cycles: 8 cycles  
bytes: 2 bytes = 1 opcode + 1 8bit operand  
operation: ACC = ACC ^ #8

This instruction performs a logical bit wise XOR operation of the current content of the accumulator with an 8 bit immediate value. the result is placed back to the accumulator.

opcode:

0	0	0	1	1	1	1	1
---	---	---	---	---	---	---	---

n	n	n	n	n	n	n	n
---	---	---	---	---	---	---	---

examples:

```
ldi    acc,#1Fh    ; ACC = 1Fh
xoracci #76h       ; ACC = E9h
```

### **CMPACCI #8**

cycles: 8 cycles

bytes: 2 bytes = 1 opcode + 1 8bit operand

operation:temp = ACC - #8

This instruction

ADDACCI

**CUSTOMIZING THE IP TO WORK FOR YOU**