8-Bit CPU

- Author: University of Waterloo - Fall 2024 ECE 298A
- Description: A basic 8-bit CPU design building off the SAP-1
- Language: Verilog

How it works

This project is a basic 8-bit CPU design building off the SAP-1. It is a combination of various modules developed as a part of the ECE298A Course at the University of Waterloo.

The control block is implemented using a 6 stage sequential counter for sequencing micro-instructions, and a LUT for corresponding op-code to operation(s).

The program counter enumerates all values between 0 and F (15) before looping back to 0 and starting again. The counter will clear back to 0 whenever the chip is reset.

The Instruction register stores the current instructions and breaks it up into the opcode and address, which are passed into corresponding locations

The 16 Byte memory module consists of 16 memory locations that store 1 byte each. The memory allows for both read and write operations, controlled by input signals, as well as data supplied by the MAR.

The MAR is a register which handles RAM interactions, namely specifying the address for store/load, as well as the data to be stored.

The 8-bit ripple carry adder assumes 2s complement inputs and thus supports addition and subtraction. It pushes the result to the bus via tri-state buffer. It also includes a zero flag and a carry flag to support conditional operation using an external microcontroller. These flags are synchronized to the rising edge of the clock and are updated when the adder outputs to the bus.

The Accumulator register functions to store the output of the adder. It is synchronized to the positive edge of the clock. The accumulator loads and outputs its value from the bus and is connected via tri-state buffer. The accumulator's current value is always available as an output (and usually connected to the Register A input of the ALU)

The B register stores the second operand for ALU operations which is loaded from RAM.

The Output register outputs the value from register A onto the uo_out pins.

The 8 Bit Bus is driven by various blocks. We allow multiple blocks that are able to write using tri-state buffers.

Supported Instructions

Mnemonic	Opcode	Function
HLT	0x0	Stop processing
NOP	0x1	No operation
ADD {address}	0x2	Add B register to A register, leaving result in A
SUB {address}	0x3	Subtract B register from A register, leaving result in A
LDA {address}	0x4	Put RAM data at {address} into A register
OUT	0x5	Put A register data into Output register and display
STA {address}	0x6	Store A register data in RAM at {address}
JMP {address}	0x7	Change PC to {address}

Instruction Notes

- All instructions consist of an opcode (most significant 4 bits), and an address (least significant 4 bits, where applicable)

Control Signal Descriptions

Control Signal	Array	Component	Function
Cp	14	PC	Increments the PC by 1
Ep	13	PC	Enable signal for PC to drive the bus
Lp	12	PC	Tells PC to load value from the bus
nLma	11	MAR	Tells MAR when to load address from the bus
nLmd	10	MAR	Tells MAR when to load memory from the bus
nCE	9	RAM	Enable signal for RAM to drive the bus
nLr	8	RAM	Tells RAM when to load memory from the MAR
nLi	7	IR	Tells IR when to load instruction from the bus
nEi	6	IR	Enable signal for IR to drive the bus
nLa	5	A Reg	Tells A register to load data from the bus
Ea	4	A Reg	Enable signal for A register to drive the bus
Su	3	ALU	Activate subtractor instead of adder
Eu	2	ALU	Enable signal for Adder/Subtractor to drive the bus

Control Signal	Array	Component	Function
nLb	1	B Reg	Tells B register to load data from the bus
nLo	0	Output Reg	Tells Output register to load data from the bus

Sequencing Details

- The control sequencer is negative edge triggered, so that control signals can be steady for the next positive clock edge, where the actions are executed.
- In each clock cycle, there can only be one source of data for the bus, however any number components can read from the bus.
- Before each run, a CLR signal is sent to the PC and the IR.

Instruction Micro-Operations

Stage	**HLT**	**NOP**	**STA**	**JMP**
T0	Ep, nLma	Ep, nLma	Ep, nLma	Ep, nLma
T1	Cp	Cp	Cp	Cp
T2	nCE, nLi	nCE, nLi	nCE, nLi	nCE, nLi
T3	**	-	nEi, nLma	nEi, Lp
T4	-	-	Ea, nLmd	
T5	-	-	nLr	

Stage	**LDA**	**ADD**	**SUB**	**OUT**
T0	Ep, nLma	Ep, nLma	Ep, nLma	Ep, nLma
T1	Cp	Cp	Cp	Cp
T2	nCE, nLi	nCE, nLi	nCE, nLi	nCE, nLi
T3	nEi, nLma	nEi, nLma	nEi, nLma	Ea, nLo
T4	nCE, nLa	nCE, nLb	nCE, nLb	-
T5	-	Eu, nLa	Su, Eu, nLa	-

Instruction Micro-Operations Notes

- First three micro-operations are common to all instructions.

- NOP operation executes only the first three micro-operations.

- Cp signal is not asserted during the HLT instruction in T2.
- ** Halt internal register is set to 1. More on this later

Programmer

Stage	**Control Signals**	**Programmer specific signals**
T0	Ep, nLMA	ready = 1
T1	Cp	ready = 0
T2	-	-
T3	nLmd	read_ui_in = 1
T4	nLr	read_ui_in = 0, done_load = 1
T5	-	done_load = 0

Detailed Overview

T0: Control Signals the same as the typical default microinstruction – load the MAR with the address of the next instruction. Assert ready signal to alert MCU programmer (off chip) that CPU is ready to accept next line of RAM data.

T1: Increment the PC, the same as the typical default microinstruction. De-assert ready signal since the MCU programmer is polling for the rising edge.

T2: Do nothing to allow an entire clock cycle for programmer to prepare the data.

T3: Load the MAR with the data from the bus. Also, assert the read_ui_in signal which controls a series of tri-state buffers, attaches the ui_in pins straight to the bus.

T4: Load the RAM from the MAR. De-assert the read_ui_in signal (disconnect the ui_in pins from driving the bus since the ui_in pin data might be now inaccurate). Assert the done_load signal to indicate to the MCU that the chip is done with the ui_in data.

T5: De-assert done_load signal.

Programmer Notes

The MCU must be able to provide the data to the ui_in pins (steady) between receiving the ready signal (assume worst case end of T0), and the bus needing the values (assume worst case beginning of T3).

Therefore, the MCU must be able to provide the data at a maximum of 2 clock periods.

IO Table: CB (Control Block)

Name	Verilog	Description	I/O	Width	Trigger
clk	clk	Clock signal	I	1	Edge Transition
resetn	rst_n	Set stage to 0	I	1	Active Low
opcode	opcode	Opcode from IR	I	4	NA
out	control_signals	Control Signal Array	O	15	NA
programming	programming	Programming mode	I	1	Active High
done_load	done_load	Executed Load during prog	O	1	Active High
read_ui_in	read_ui_in	Push ui_in onto bus	O	1	Active High
ready	ready_for_ui	Ready to prog next byte	O	1	Active High
HF	HF	Halting flag	O	1	Active High

IO Table: PC (Program Counter)

Name	Verilog	Description	I/O	Width	Trigger
bus	bus[3:0]	Connection to bus	IO	4	NA
clk	clk	Clock signal	I	1	Falling Edge
clr_n	rst_n	Clear to 0	I	1	Active Low
cp	Ep	Allow counter increment	I	1	Active High
ep	Cp	Output to bus	I	1	Active High
lp	Lp	Load from bus	I	1	Active High

PC (Program Counter) Notes

- Counter increments only when Cp is asserted, otherwise it will stay at the current value.

- Ep controls whether the counter is being output to the bus. If this signal is low, our output is high impedance (Tri-State Buffers).
- When CLR is low, the counter is cleared back to 0, the program will restart.
- The program counter updates its value on the falling edge of the clock.
- Lp indicates that we want to load the value on the bus into the counter (used for jump instructions). When this is asserted, we will read from the bus and instead of incrementing the counter, we will update each flip-flop with the appropriate bit and prepare to output.
- The least significant 4 bits from the 8-bit bus will be used to store the value on the program counter (0-15). Will be read from (JMP asserted) and written to (Ep asserted).
- clr_n has precedence over all.
- Lp takes precedence over Cp.

IO Table: Instruction Register (IR)

Name	Verilog	Description	I/O	Width	Trigger
bus	bus	Connection to bus	IO	8	NA
clk	clk	Clock signal	I	1	Rising Edge
clear	~rst_n	Clear to 0	I	1	Active High
opcode	opcode	Opcode from IR	O	4	NA
n_load	nLi	Load from Bus	I	1	Active Low
n_enable	nEi	Output to bus	O	1	Active Low

Instruction Register (IR) Notes

- The A Register updates its value on the rising edge of the clock.
- nEi controls whether the instruction is being output to the bus[3:0]. If this signal is high, our output is high impedance (Tri-State Buffers).
- nLi indicates that we want to load the value on the bus into the IR. When this is low, we will read from the bus and write to the register.
- When clear is high, the opcode is cleared back to NOP.
- IR always outputs the current value of the register to CB.

IO Table: RAM

Name	Verilog	Description	I/O	Width	Trigger
addr	mar_to_ram_addr	Address for read/write	I	4	NA
data_in	mar_to_ram_data	Data for write	I	8	NA
data_out	bus	Connection to bus	O	8	NA
lr_n	nLr	Load data from MAR	I	1	Active Low
ce_n	nCE	Output to bus	I	1	Active Low
clk	clk	Clock Signal	I	1	Rising edge
rst_n	'1'	Clear RAM	I	1	Active Low

RAM Notes

- Addressing: The memory is 4-bit addressable, where the address specifies which register (out of 16) is being accessed for reading or writing.
- Write operation: A byte of data is written to specific register in RAM, where the location is determined by the address. Requires write enable lr_n signal as active (low) and the clock edge to occur.
- Read operation: Data can be read from a specific register in RAM determined by the input address. Requires chip enable ce_n signal as active (low). The data is output on the bus, and it is updated on the clock edge.
- Output: Data is presented on the bus line when the chip is enabled for reading, and high-impedance (Z) otherwise.
- RAM is never reset, rather, we always flash it.

IO Table: MAR

Name	Verilog	Description	I/O	Width	Trigger
bus	bus	Connection to bus	IO	8	NA
clk	clk	Clock signal	I	1	Rising Edge
addr	mar_to_ram_addr	Address for read/write	O	4	NA
data	mar_to_ram_data	Data for write	O	8	NA
n_load_data	nLmd	Load data from Bus	I	1	Active Low
n_load_addr	nLma	Load address from Bus	I	1	Active Low

MAR Notes

- The MAR updates its value on the rising edge of the clock.

- nLmd indicates that we want to load the value on the bus into the data register. When this is low, we will read from the bus and write to the register.
- nLma indicates that we want to load the value on the bus[3:0] into the address register. When this is low, we will read from the bus and write to the register.
- MAR always outputs the current value of the data and address registers to the RAM module.

IO Table: ALU (Adder/Subtractor)

Name	Verilog	Description	I/O	Width	Trigger
clk	clk	Clock Signal	I	1	Rising edge
enable_out	Eu	Output to bus	I	1	Active High
Register A	reg_a	Accumulator Register	I	8	NA
Register B	reg_b	Register B	I	8	NA
subtract	sub	Perform Subtraction	I	1	Active High
bus	bus	Connection to bus	O	8	NA
Carry Out	CF	Carry-out flag	O	1	Active High
Result Zero	ZF	Zero flag	O	1	Active High

ALU (Adder/Subtractor) Notes

- Eu controls whether the counter is being output to the bus. If this signal is low, our output is high impedance (Tri-State Buffers).
- A Register and B Register always provide the ALU with their current values.
- When sub is not asserted, the ALU will perform addition: Result = A + B
- When sub is asserted, the ALU will perform subtraction by taking 2s complement of operand B: Result = A - B = A + !B + 1
- Carry Out and Result Zero flags are updated on rising clock edge.

IO Table: Accumulator (A) Register

Name	Verilog	Description	I/O	Width	Trigger
clk	clk	Clock Signal	I	1	Rising edge
bus	bus	Connection to bus	IO	8	NA
load	nLa	Load from bus	I	1	Active Low
enable_out	Ea	Output to bus	I	1	Active High

Name	Verilog	Description	I/O	Width	Trigger
Register A	reg_a	Accumulator Register	O	8	NA
clear	rst_n	Clear Signal	I	1	Active Low

Accumulator (A) Register Notes

- The A Register updates its value on the rising edge of the clock.
- Ea controls whether the counter is being output to the bus. If this signal is low, our output is high impedance (Tri-State Buffers).
- nLa indicates that we want to load the value on the bus into the A Register. When this is low, we will read from the bus and write to the register.
- When CLR is low, the register is cleared back to 0.
- (Register A) always outputs the current value of the register to the ALU.

IO Table: B Register

Name	Verilog	Description	I/O	Width	Trigger
bus	bus	Connection to bus	IO	8	NA
clk	clk	Clock Signal	I	1	Rising edge
n_load	nLb	Load from bus	I	1	Active Low
value	reg_b	B Register value	O	8	NA

B Register Notes

- The B Register updates its value on the rising edge of the clock.
- nLb indicates that we want to load the value on the bus into the B Register. When this is low, we will read from the bus and write to the register.
- B Register always outputs the current value of the register to the ALU.

IO Table: Output Register

Name	Verilog	Description	I/O	Width	Trigger
bus	bus	Connection to bus	IO	8	NA
clk	clk	Clock Signal	I	1	Rising edge

Name	Verilog	Description	I/O	Width	Trigger
n_load	nLo	Load from bus	I	1	Active Low
value	uo_out	B Register value	O	8	NA

Output Register Notes

- The Output Register updates its value on the rising edge of the clock.
- nLo indicates that we want to load the value on the bus into the B Register. When this is low, we will read from the bus and write to the register.

How to test

Provide input of op-code. Check that the correct output bits are being asserted/de-asserted properly.

Setup

1. **Power Supply**: Connect the chip to a stable power supply as per the voltage specifications.
2. **Clock Signal**: Provide a stable clock signal to the `clk` pin.
3. **Reset**: Ensure the `rst_n` pin is properly connected to allow resetting the chip.

Testing Steps

1. **Initial Reset**:

   - Perform a sync reset by pulling the `rst_n` pin low, waiting for 1 clock signal, and then pulling pulling the `rst_n` high to initialize the chip.

2. **Load Program into RAM**:

   - Use the `ui_in` pins to load a test program into the RAM. Ensure the `programming` pin is high during this process.
   - Perform a sync reset by pulling the `rst_n` pin low, waiting for 1 clock signal, and then pulling pulling the `rst_n` high to initialize the chip.
   - Wait for the ready_for_ui signal to go high, indicating that the CPU is ready to accept data.

- Provide the first byte of data on the ui_in pins.
- Wait for the done_load signal to go high, indicating that the data has been successfully loaded into the RAM.
- Repeat the process for each byte of data:
  - Wait for ready_for_ui to go high.
  - Provide the next byte of data on the ui_in pins.
  - Wait for done_load to go high.
- Example program data:

```
0x10,  # NOP
0x73,  # JMP 0x3
0x00,  # HLT
0x4F,  # LDA 0xF
0x2E,  # ADD 0xE
0x6D,  # STA 0xD
0x50,  # OUT
0x3F,  # SUB 0xF
0x50,  # OUT
0x4D,  # LDA 0xD
0x50,  # OUT
0x72,  # JMP 0x2
0x10,  # NOP
0x00,  # Padding/empty instruction
0x02,  # Constant 2 (data)
0x01   # Constant 1 (data)
```

3. **Run Test Program**:

- Set the `programming` pin low to exit programming mode.
- Perform a sync reset by pulling the `rst_n` pin low, waiting for 1 clock signal, and then pulling pulling the `rst_n` high to initialize the chip.
- Monitor the `uo_out` and `uio_out` pins for expected outputs.
- Verify the control signals and data outputs at each clock cycle.

4. **Functional Tests**:

- Perform specific functional tests for each instruction (e.g., ADD, SUB, LDA, STA, JMP, HLT).
- Verify the correct execution of each instruction by checking the output and control signals.

Example Test Cases

- **HLT Instruction**: Example program data:

```
0x4E,  # LDA 0xE
0x50,  # OUT
0x00,  # HLT
0x4F,  # LDA 0xF
0x50,  # OUT
0x00,  # HLT
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x00,  # Padding/empty instruction
0x09,  # Constant 9 (data)
0xFF   # Constant 255 (data)
```

  This program should first output 9 and then NOT change that to 255. HF should be set to 1

- **NOP Instruction**: Example program data:

```
0x42,  # LDA 0x2
0x50,  # OUT
0x10,  # NOP / Constant 16 (data)
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x4E,  # LDA 0xF
0x50,  # OUT
0x1F,  # NOP
0x1F,  # NOP / Constant 31 (data)
```

This program should flash the lower 4 bits of the output register on and off with different on/off times

- **NOP Instruction**: Example program data:

```
0x42,  # LDA 0x2
0x50,  # OUT
0x10,  # NOP / Constant 16 (data)
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x1F,  # NOP
0x4E,  # LDA 0xF
0x50,  # OUT
0x1F,  # NOP
0x1F,  # NOP / Constant 31 (data)
```

This program should flash the lower 4 bits of the output register on and off with different on/off times

- **ADD Instruction** Example program data:

```
0x50,  # OUT
0x2E,  # ADD 0xE
0x70,  # JMP 0x0
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0x01,  # Constant 1 (data)
0xFF,  # Padding/empty instruction
```

This program should add 1 to the A register, display it and loop back to the start. The output should be a counter from 0 to 255, then repeat.

CF should be set to 1 when the A register overflows, and 0 when it doesn't. CF=1 happens when the A register is 255 and 1 is added to it.

ZF should be set to 1 when the A register is 0, and 0 otherwise.

- **SUB Instruction** Example program data:

```
0x50,  # OUT
0x3E,  # SUB 0xE
0x70,  # JMP 0x0
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0x01,  # Constant 1 (data)
0xFF,  # Padding/empty instruction
```

  This program should subtract 1 to the A register, display it and loop back to the start. The output should be a counter from 255 to 0, then repeat.

  CF should be set to 1 when the A register overflows, and 0 when it doesn't. CF=0 happens when the A register is 0 and 1 is subtracted from it.

  ZF should be set to 1 when the A register is 0, and 0 otherwise.

- **LDA Instruction**

  See above for example program data.

- **OUT Instruction**

  See above for example program data.

- **STA Instruction**

  Example program data:

```
0x4E,  # LDA 0xE
0x2F,  # ADD 0xF
0x5F,  # OUT
0x6E,  # STA 0xF
0x2F,  # ADD 0xE
0x5F,  # OUT
0x00,  # HLT
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0x09,  # Constant 9 (data)
0xFF   # Constant 255 (data) -> Constant 8 (data)
```

This program should load 9 to the A register, add 255 to it, resulting in 8 (CF should set to 1) display it, store it in 0xF, add 9 to it, resulting in 17 (CF should set to 0) and display it. Then, it should halt, and set HF to 1.

- **JMP Instruction**

  Example program data:

```
0x44,  # LDA 0x4
0x5F   # OUT
0x7D,  # JMP 0xD
0x0F,  # HLT
0x00,  # Constant 0 (data)
0xFF,  # Constant 5 (data)
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0xFF,  # Padding/empty instruction
0x45,  # LDA 0x5
0x5F   # OUT
0x0F,  # HLT
```

This program should load 0x4 (0) to the A register, display it, NOT HALT, jump to 0xD, then load 0x5 (255) to the A register, display it, and halt. HF should be set to 1.

Acknowledgements

- Darius Rudaitis, Eshann Mehta: RAM
- Evan Armoogan, Catherine Ye: PC
- Damir Gazizullin, Owen Golden: ALU, Accumulator
- Roni Kant, Jeremy Kam: MAR, B Register, Output Register, Instruction Register
- Gerry Chen, Siddharth Nema: Control Block and Programmer
- ECE 298A Course Staff: Prof. John Long, Prof. Vincent Gaudet, Refik Yalcin

Pinout

#	Input	Output	Bidirectional
0	prog_in_0	output_register_0	in_programming
1	prog_in_1	output_register_1	out_ready_for_ui
2	prog_in_2	output_register_2	out_done_load
3	prog_in_3	output_register_3	out_CF
4	prog_in_4	output_register_4	out_ZF
5	prog_in_5	output_register_5	out_HF
6	prog_in_6	output_register_6	
7	prog_in_7	output_register_7	