Designing the processor

Datapath and Control Signals

- Click here

Instruction Set

There are 31 instructions that can be used by the assembly programmer. Some instructions have a 16-bit operand, and most instructions are zero operand instructions. 8 bits are used for opcode. Following table explains the operations of each instruction.

Instruction	Opcode	Operand (16 bit)	Operation
loadac	0000 0100	-	ac <= DM[ac]
movacr	0000 1000	-	r <= ac
movacr1	0000 1001	-	r1 <= ac
movacr2	0000 1010	-	r2 <= ac
movacr3	0000 1011	-	r3 <= ac
movacr4	0000 1100	-	r4 <= ac
movacr5	0000 1101	-	r5 <= ac
movacdar	0000 1110	-	dar <= ac
movrac	0000 1111	-	ac <= r
movr1ac	0001 0000	-	ac <= r1
movr2ac	0001 0001	-	ac <= r2
movr3ac	0001 0010	-	ac <= r3
movr4ac	0001 0011	-	ac <= r4
movr5ac	0001 0100	-	ac <= r5
movdarac	0001 0101	-	ac <= dar
stac	0001 0110	-	DM[ac] <= ac
add	0001 1001	-	ac <= ac + r
sub	0001 1011	-	ac <= ac – r
lshift	0001 1101	-	ac <= r << ac
rshift	0001 1111	-	ac <= r >> ac
incac	0010 0001	-	ac <= ac + 1
incdar	0010 0010	-	dar <= dar +1
incr1	0010 0011	-	r1 <= r1 +1
incr2	0010 0100	-	r2 <= r2 +1
incr3	0010 0101	-	r3 <= r3 + 1
loadim	0010 0110	imm	ac <= imm
jumpz	0010 1001	imm	if z == 1, go to instruction IM[imm]
jumpnz	0011 0000	imm	if z == 0, go to instruction IM[imm]
jump	0011 0001	imm	go to instruction IM[imm]
nop	0011 0010	-	no operation
endop	0011 0011	-	end process

Microinstructions

These microinstructions are used for finite state machine in the control unit. one microinstruction can be operated in one clock cycle. Following table explains operations of each microinstruction.

Instruction (8bits)	Opcode (decimal)	Microinstruction (6bits)	Operation
idle (internal)	-	idle	goto fetch1 if status == 1 else idle
fetch (internal)	-	fetch1	read IMEM
			write ir
			goto fetch2
		fetch2	read IMEM
			write ir
			goto fetch3
		fetch3	read IMEM
			goto ir[5:0]
loadac	4	loadac1	read ac
			write dar
			goto loadac2
		loadac2	read ac
			wrirte dar
			goto loadac3
		loadac3	read DMEM
			write ac
			goto loadac4
		loadac4	read DMEM
			write ac
			increment pc
			goto fetch1
movacr	8	movacr	read ac
			write r
			increment pc
			goto fetch1
movacr1	9	moveacr1	read ac
			write r1
			increment pc
			goto fetch1
movacr2	10	moveacr2	read ac
			write r2
			increment pc
			goto fetch1
movacr3	11	moveacr3	read ac
			write r3
			increment pc
			goto fetch1
movacr4	12	moveacr4	read ac

			write r4
			increment pc
			goto fetch1
movacr5	13	moveacr5	read ac
			write r5
			increment pc
			goto fetch1
movacdar	14	moveacdar	read ac
			write dar
			increment pc
			goto fetch1
movrac	15	moverac	read r
			write ac
			increment pc
			goto fetch1
movr1ac	16	mover1ac	read r1
			write ac
			increment pc
			goto fetch1
movr2ac	17	mover2ac	read r2
			write ac
			increment pc
			goto fetch1
movr3ac	18	mover3ac	read r3
			write ac
			increment pc
			goto fetch1
movr4ac	19	mover4ac	read r4
			write ac
			increment pc
			goto fetch1
movr5ac	20	mover5ac	read r5
			write ac
			increment pc
			goto fetch1
movdarac	21	movedarac	read dar
			write ac
			increment pc
			goto fetch1
stac	22	stac1	read ac
			goto stac2
		stac2	read ac
			write dm
			goto stac3
		stac3	read ac
			increment pc
			goto fetch1

add	25	add1	add alu
			goto add2
		add2	add alu
			write alu_to_ac
			increment pc
			goto fetch1
sub	27	sub1	sub alu
			goto sub2
		sub2	sub alu
			write alu_to_ac
			increment pc
			goto fetch1
lshift	29	lshift1	lshift alu
			goto lshift2
		lshift2	lshift alu
			write alu_to_ac
			increment pc
			goto fetch1
rshift	31	rshift1	rshift alu
			goto rshift2
		rshift2	rshift alu
			write alu_to_ac
			increment pc
			goto fetch1
incac	33	incac	increment ac
			incremtnt pc
			goto fetch1
incdar	34	incdar	increment dar
			incremtnt pc
			goto fetch1
incr1	35	incr1	increment r1
			incremtnt pc
			goto fetch1
incr2	36	incr2	increment r2
			incremtnt pc
			goto fetch1
incr3	37	incr3	increment r3
			incremtnt pc
			goto fetch1
loadim	38	loadim1	increment pc
			goto loadim2
		loadim2	read IMEM
			goto loadim3
		loadim3	read IMEM
			write ac
			increment pc
			goto fetch1

jump	49	jump	increment pc
			goto jumpz2
jumpz	41	jumpz1	increment pc
			goto jumpz2 if z == 1 else jumpz6
		jumpz2	read IMEM
			write pc
			goto jumpz3
		jumpz3	read IMEM
			write pc
			goto jumpz4
		jumpz4	read IMEM
			write ir
			goto jumpz5
		jumpz5	read IMEM
			write ir
			goto fetch3
		jumpz6	goto jumpz7
		jumpz7	increment pc
			goto jumpz4
jumpnz	48	jumpnz1	increment pc
			goto jumpz3 if z == 0 else goto jumpz7
endop	51	endop	read DMEM
			goto endop
nop	50	nop	increment pc
			goto fetch1

Finite State Machines for Microinstructions

- Click here

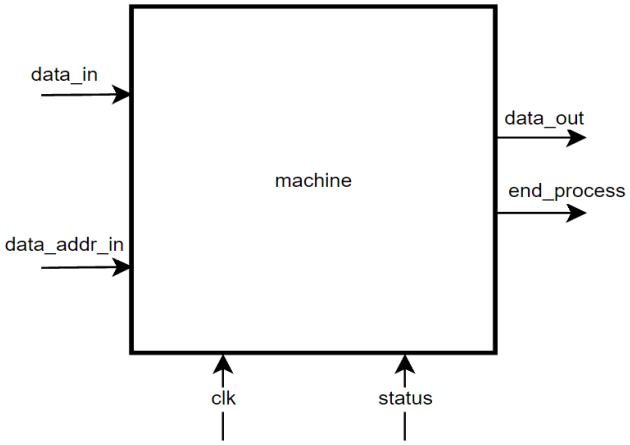
Modules and Components

Machine

There are three phases in this machine.

1. Load the image – Phase 1
2. Processing – Phase 2
3. Get the processed image – Phase 3

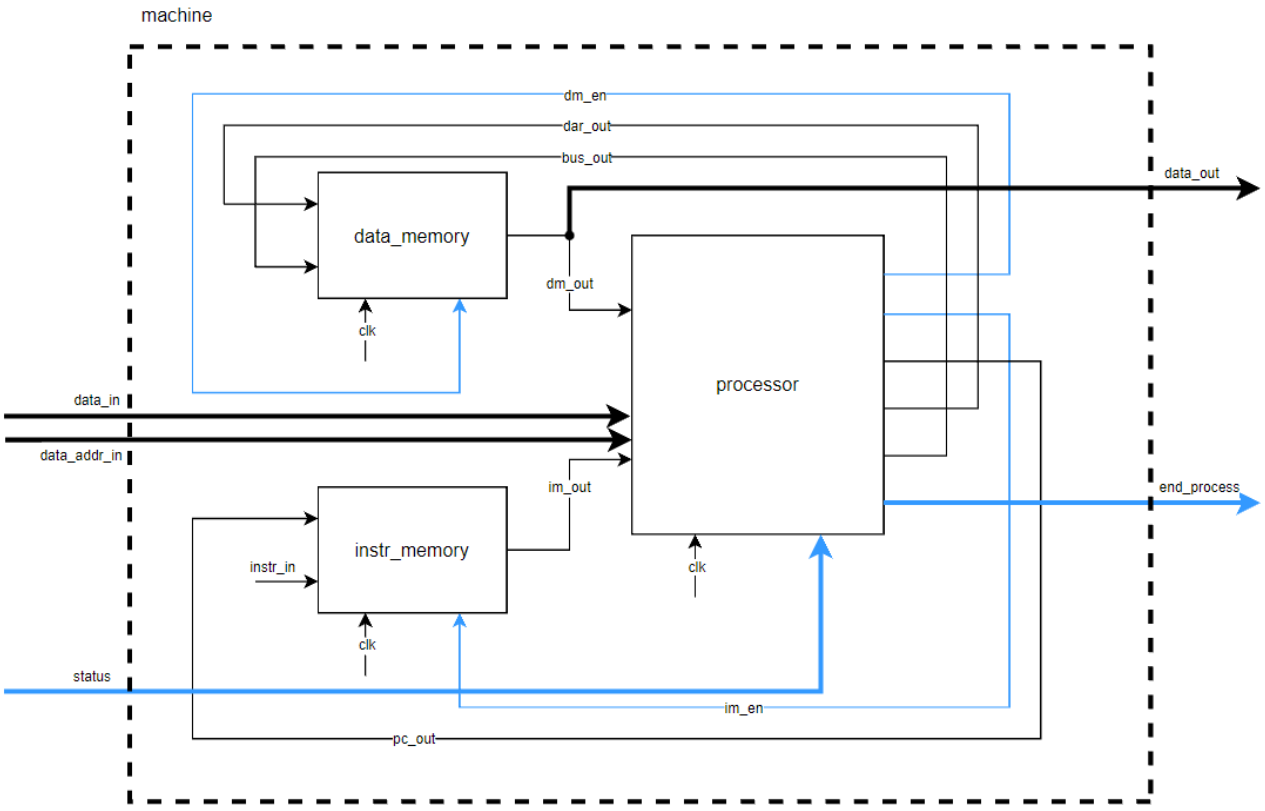
In phase 1, we have to give the image data to data_in and memory address to data_addr_in. Then processing will be done. After phase 2 is done, end_process will be asserted. In phase 3, we can get the output image data from the data_out.



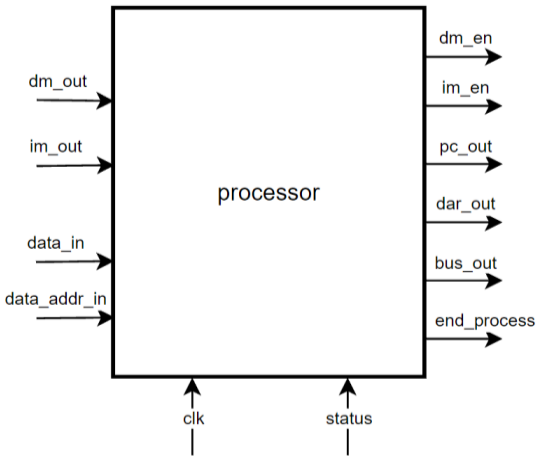
Inputs	Outputs
clk – clock	**end_process** – signal that indicate process has ended
status – control signal for change phases (input image data, process data, output data)	**data_out** – ouput image data
data_in – input image data	
data_addr_in – input data memory address	

status	10	01	11
phase	1 – load image	2 – process	3 – output image

Inside of the machine



Processor



Inputs	Outputs
clk – clock	**end_process** – signal that indicate process has ended
status – control signal for change phases (input image data, process data, output data)	**dm_en** – data memory write enable signal
dm_out – output of data memory	**im_en** – instruction memory write enable signal
im_out – output of instruction memory	**pc_out** – instruction memory address
data_in – input image data	**dar_out** – data memory address
data_addr_in – input data memory address	**bus_out** – data output for data memory

- Inside of the Processor – Click here

Modules Inside the Processor

Arithmetic & Logic Unit (ALU) – Do arithmetic and logic operations in the processor

Control Unit – Do all the controlling in the processor. Implemented as a finite state machine.

Special purpose registers

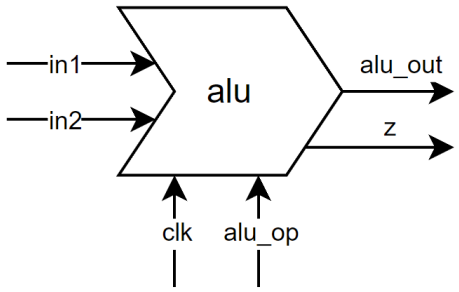
- Accumulator (ac) – input for the ALU, store ALU output
- Program counter (pc) – directed to the next instruction to be fetched
- Instruction register (ir) – store the instruction fetched from the instruction memory
- Data address register (dar) – keep the address of the store location of the data memory. Loading from data memory and storing to data memory are being done using this address
- r register – one of the inputs for the ALU

General purpose registers

- r1, r2, r3 – general purpose registers which can be incremented
- r4, r5 – general purpose registers which cannot be incremented

ALU

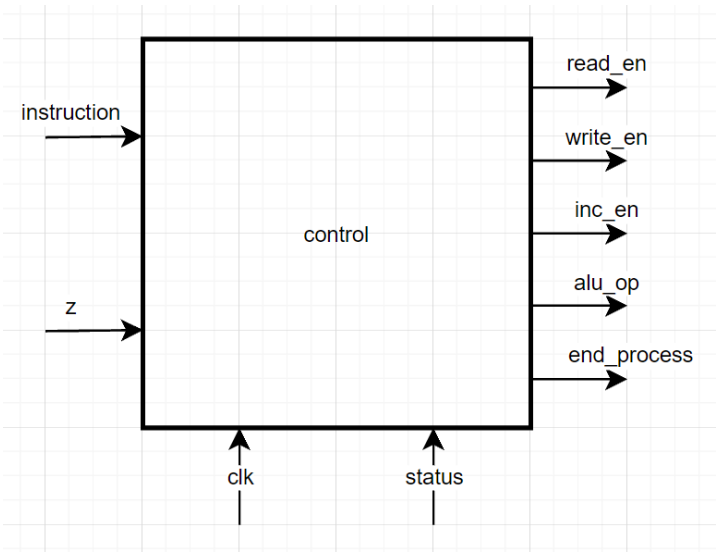
ALU is responsible for all the arithmetic and logic operations. Inputs are taken from ac and r registers and output is stored in ac register. We are using 3-bit alu_op even though there are only four operations since it is better to have some improvement capabilities. z will be 1 if the alu_out is zero. z will be 0 if the alu_out is nonzero.



alu_op	Operation	Description
001	Addition	Alu_out <= in1 + in2
010	Subtraction	Alu_out <= in2 – in1
011	Left shift	Alu_out <= in1 << in2
100	Right shift	Alu_out <= in1 >> in2

Control Unit

Control Unit is responsible for all the control signals in the processor. Here we are implemented this using a finite state machine.



Inputs	Outputs
clk – clock	**end_process** – signal that indicate process has ended
status – control signal for change phases (input image data, process data, output data)	**read_en** – 4 bit read enable signal
instruction – instruction from the instruction register	**write_en** – 16 bit write enable signal
z – z output from the alu	**inc_en** – 8 bit increment enable signal
	alu_op – ALU opcode

Read Enable – read_en

read_en is the control signal that is responsible for all the readings in the processor. Read values are written to the data bus. We cannot read from multiple registers and write them to the same bus. Hence, we are using 4 bit signal such that only one register can be read at one time.

Decimal Value	Register
0	-
1	dar
2	ac
3	r
4	r1
5	r2
6	r3
7	r4
8	r5
9	DM
10	IM

Write Enable – write_en

write_en is the control signal that is responsible for all the writings in the processor. Unlike the reading, we can write data to multiple locations at the same time. Therefore, we are using 16-bit signal and each bit has assigned to separate location. We can write those location by setting those bits to 1.

Bit 15 N/A	Bit 14 N/A	Bit 13 alu to ac	Bit 12 IM	Bit 11 DM	Bit 10 r1	Bit 9 r2	Bit 8 r3	Bit 7 r4	Bit 6 r5	Bit 5 r	Bit 4 ac	Bit 3 ir	Bit 2 dar	Bit 1 pc	Bit 0 N/A

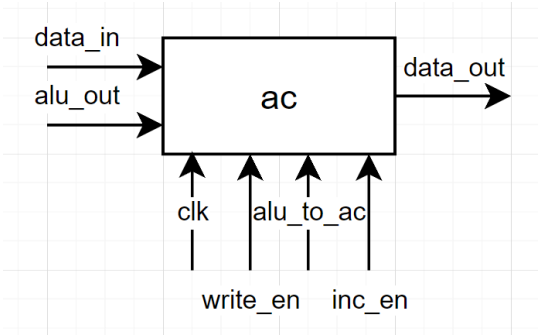
Increment Enable – inc_en

inc_en is the control signal that is responsible for all the increments in the processor. Same as the writing, we can increment multiple registers at the same time.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
N/A	r3	r2	r1	dar	ac	pc	N/A

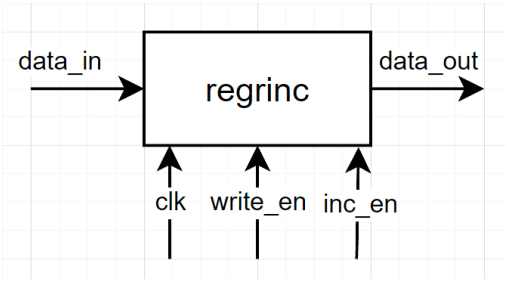
Accumulator

Accumulator is a special purpose register which has two data inputs. One is from the data bus and the other one is from the ALU output. There are two write enable signals to indicate from which input the writing should be done. This is one of the inputs to the ALU.



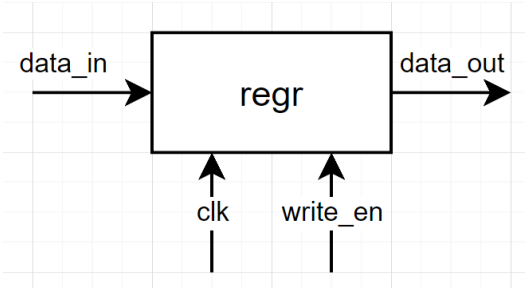
Register Which Can Be Incremented

pc, dar, r1, r2, r3 are registers of this type. The values in those registers can be incremented without using the ALU.



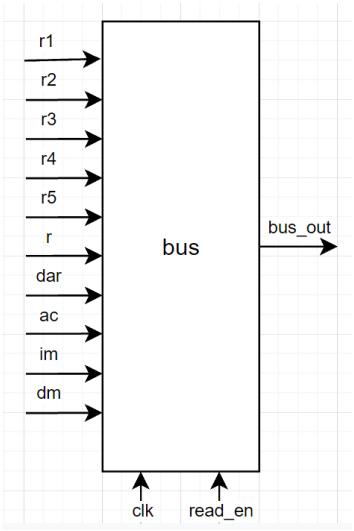
Register Which Cannot Be Incremented

ir, r, r4, r5 are registers of this type. The values in those registers cannot be incremented without using the ALU.



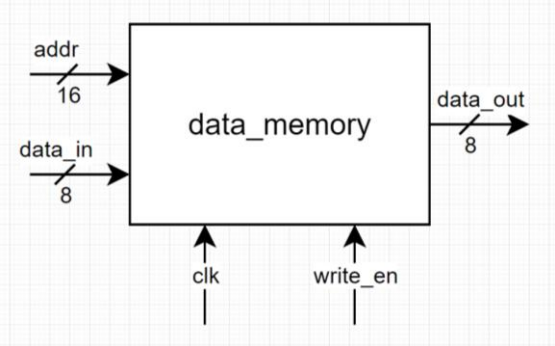
Data Bus

This is a 16-bit path which connects all the necessary components that needed to the data flow.



Data Memory

This is the main memory which data is stored. The image should be stored here before the processing begins. This memory contains 65536 locations of 8 bits long. Since we are using 256*256 image, 65536 pixel values are needed to be stored and that is why 65536 locations. Since the pixel value range is 0-255, we need 8-bit long memory to store that.



Instruction Memory

This is for storing the instructions. This is a read only memory. The processor cannot write into instruction memory during the processing. This can be written only when we need to load a new programme.