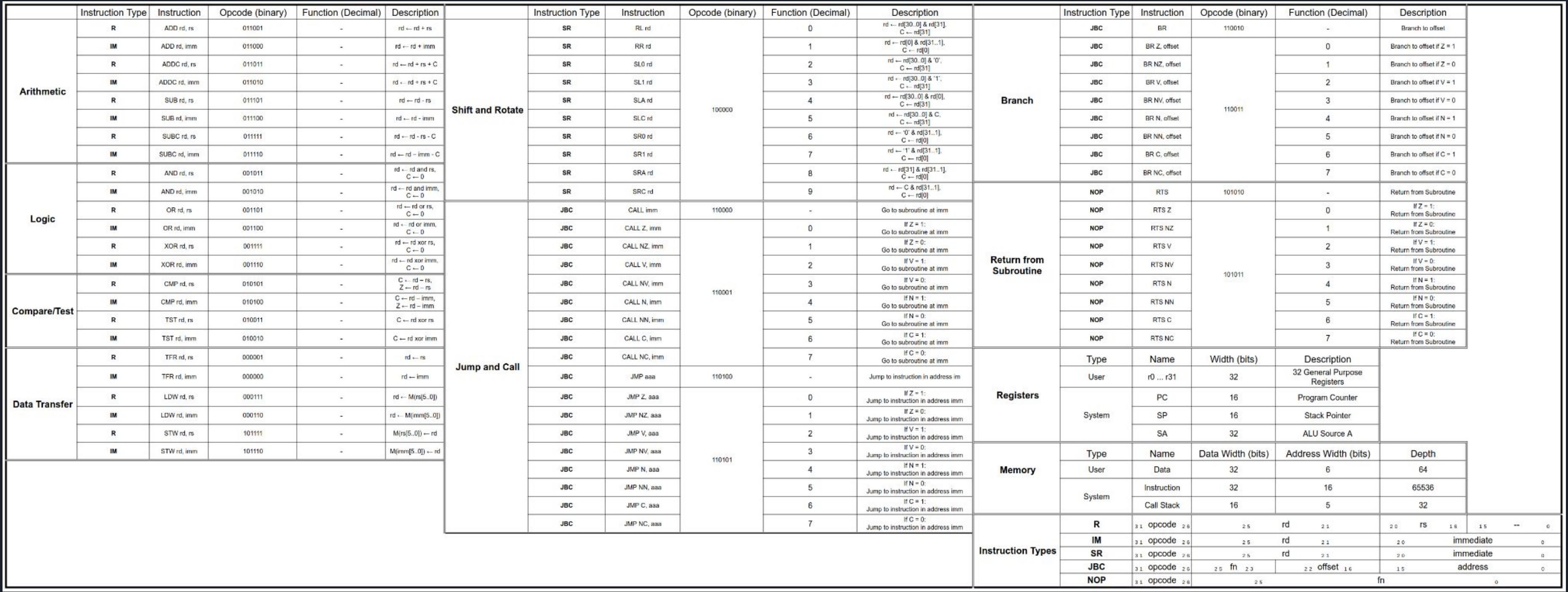
32-bit Microprocessor

Lukas Popovic, Cameron Vogeli

General Description



	Instruction Type	Instruction	Opcode (binary)	Function (Decimal)	Description
Arithmetic	R	ADD rd, rs	011001	-	rd ← rd + rs
	IM	ADD rd, imm	011000	-	rd ← rd + imm
	R	ADDC rd, rs	011011	-	rd ← rd + rs + C
	IM	ADDC rd, imm	011010	-	rd ← rd + rs + C
	R	SUB rd, rs	011101	-	rd ← rd - rs
	IM	SUB rd, imm	011100	-	rd ← rd - imm
	R	SUBC rd, rs	011111	-	rd ← rd - rs - C
	IM	SUBC rd, imm	011110	-	rd ← rd – imm - C
Logic	R	AND rd, rs	001011	-	rd ← rd and rs, C ← 0
	IM	AND rd, imm	001010	-	rd ← rd and imm, C ← 0
	R	OR rd, rs	001101	-	rd ← rd or rs, C ← 0
	IM	OR rd, imm	001100	-	rd ← rd or imm, C ← 0
	R	XOR rd, rs	001111	-	rd ← rd xor rs, C ← 0
	IM	XOR rd, imm	001110	-	rd ← rd xor imm, C ← 0
Compare/Test	R	CMP rd, rs	010101	-	C ← rd – rs, Z ← rd – rs
	IM	CMP rd, imm	010100	-	C ← rd – imm, Z ← rd – imm
	R	TST rd, rs	010011	-	C ← rd xor rs
	IM	TST rd, imm	010010	-	C ← rd xor imm
Data Transfer	R	TFR rd, rs	000001	-	rd ← rs
	IM	TFR rd, imm	000000	-	rd ← imm
	R	LDW rd, rs	000111	-	rd ← M(rs[5..0])
	IM	LDW rd, imm	000110	-	rd ← M(imm[5..0])
	R	STW rd, rs	101111	-	M(rs[5..0]) ← rd
	IM	STW rd, imm	101110	-	M(imm[5..0]) ← rd

	Instruction Type	Instruction	Opcode (binary)	Function (Decimal)	Description
Shift and Rotate	SR	RL rd	100000	0	rd ← rd[30..0] & rd[31], C ← rd[31]
	SR	RR rd		1	rd ← rd[0] & rd[31..1], C ← rd[0]
	SR	SL0 rd		2	rd ← rd[30..0] & '0', C ← rd[31]
	SR	SL1 rd		3	rd ← rd[30..0] & '1', C ← rd[31]
	SR	SLA rd		4	rd ← rd[30..0] & rd[0], C ← rd[31]
	SR	SLC rd		5	rd ← rd[30..0] & C, C ← rd[31]
	SR	SR0 rd		6	rd ← '0' & rd[31..1], C ← rd[0]
	SR	SR1 rd		7	rd ← '1' & rd[31..1], C ← rd[0]
	SR	SRA rd		8	rd ← rd[31] & rd[31..1], C ← rd[0]
	SR	SRC rd		9	rd ← C & rd[31..1], C ← rd[0]
Jump and Call	JBC	CALL imm	110000	-	Go to subroutine at imm
	JBC	CALL Z, imm	110001	0	If Z = 1: Go to subroutine at imm
	JBC	CALL NZ, imm		1	If Z = 0: Go to subroutine at imm
	JBC	CALL V, imm		2	If V = 1: Go to subroutine at imm
	JBC	CALL NV, imm		3	If V = 0: Go to subroutine at imm
	JBC	CALL N, imm		4	If N = 1: Go to subroutine at imm
	JBC	CALL NN, imm		5	If N = 0: Go to subroutine at imm
	JBC	CALL C, imm		6	If C = 1: Go to subroutine at imm
	JBC	CALL NC, imm		7	If C = 0: Go to subroutine at imm
	JBC	JMP aaa	110100	-	Jump to instruction in address im
	JBC	JMP Z, aaa	110101	0	If Z = 1: Jump to instruction in address imm
	JBC	JMP NZ, aaa		1	If Z = 0: Jump to instruction in address imm
	JBC	JMP V, aaa		2	If V = 1: Jump to instruction in address imm
	JBC	JMP NV, aaa		3	If V = 0: Jump to instruction in address imm
	JBC	JMP N, aaa		4	If N = 1: Jump to instruction in address imm
	JBC	JMP NN, aaa		5	If N = 0: Jump to instruction in address imm
	JBC	JMP C, aaa		6	If C = 1: Jump to instruction in address imm
	JBC	JMP NC, aaa		7	If C = 0: Jump to instruction in address imm

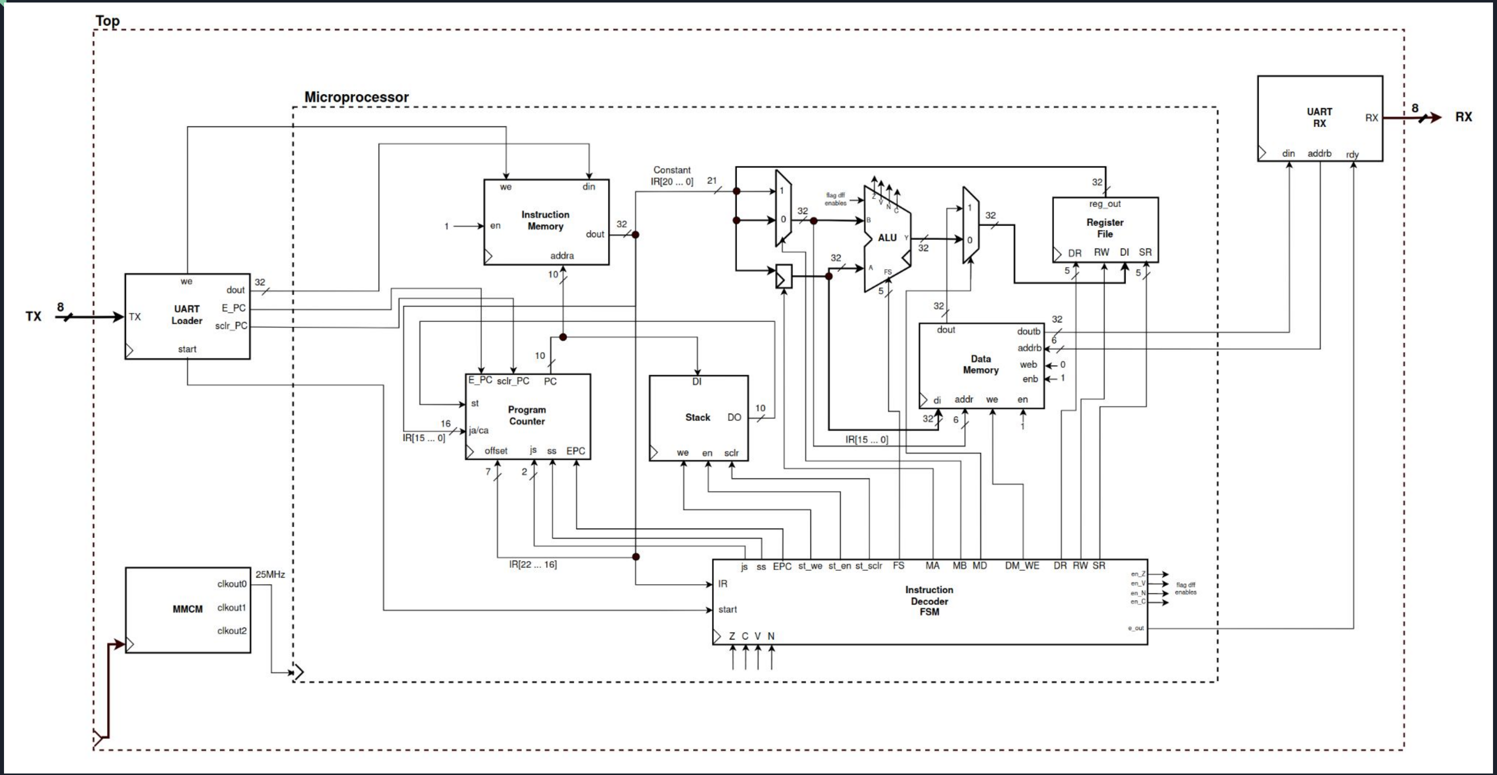
	Instruction Type	Instruction	Opcode (binary)	Function (Decimal)	Description
Branch	JBC	BR	110010	-	Branch to offset
	JBC	BR Z, offset	110011	0	Branch to offset if Z = 1
	JBC	BR NZ, offset		1	Branch to offset if Z = 0
	JBC	BR V, offset		2	Branch to offset if V = 1
	JBC	BR NV, offset		3	Branch to offset if V = 0
	JBC	BR N, offset		4	Branch to offset if N = 1
	JBC	BR NN, offset		5	Branch to offset if N = 0
	JBC	BR C, offset		6	Branch to offset if C = 1
	JBC	BR NC, offset		7	Branch to offset if C = 0
Return from Subroutine	NOP	RTS	101010	-	Return from Subroutine
	NOP	RTS Z	101011	0	If Z = 1: Return from Subroutine
	NOP	RTS NZ		1	If Z = 0: Return from Subroutine
	NOP	RTS V		2	If V = 1: Return from Subroutine
	NOP	RTS NV		3	If V = 0: Return from Subroutine
	NOP	RTS N		4	If N = 1: Return from Subroutine
	NOP	RTS NN		5	If N = 0: Return from Subroutine
	NOP	RTS C		6	If C = 1: Return from Subroutine
	NOP	RTS NC		7	If C = 0: Return from Subroutine

	Type	Name	Width (bits)	Description
Registers	User	r0 ... r31	32	32 General Purpose Registers
	System	PC	16	Program Counter
		SP	16	Stack Pointer
		SA	32	ALU Source A

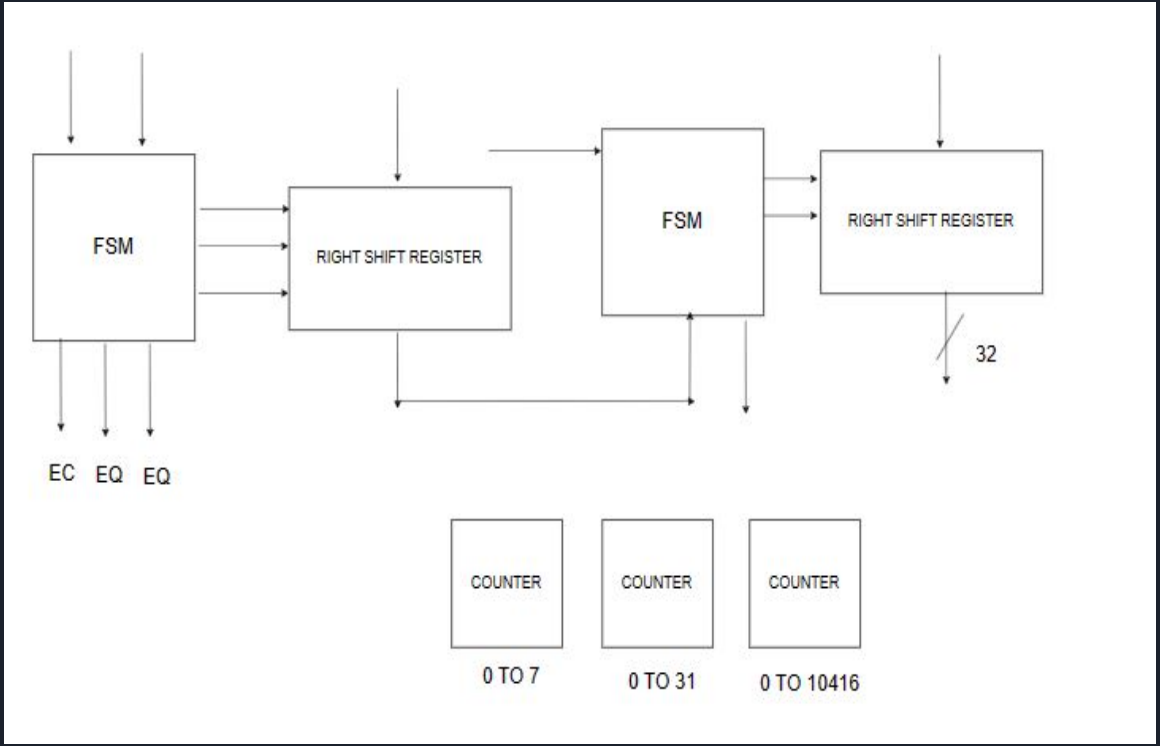
	Type	Name	Data Width (bits)	Address Width (bits)	Depth
Memory	User	Data	32	6	64
	System	Instruction	32	16	65536
		Call Stack	16	5	32

Instruction Types	R	31 opcode 26	25 rd 21	20 rs 16	15 -- 0
	IM	31 opcode 26	25 rd 21	20 immediate 0	
	SR	31 opcode 26	25 rd 21	20 immediate 0	
	JBC	31 opcode 26	25 fn 23	22 offset 16	15 address 0
	NOP	31 opcode 26	25 fn 0		

Circuit Diagram



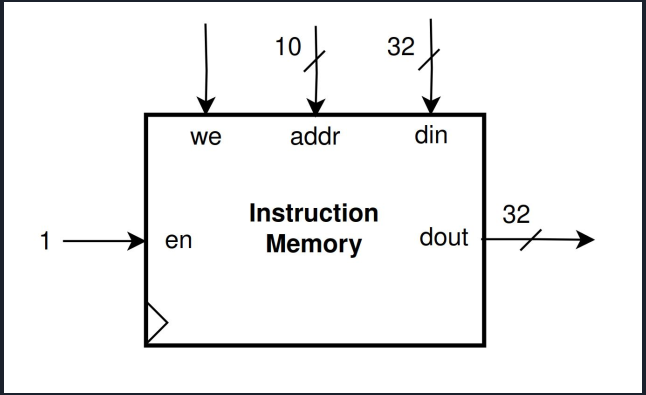
Loader


FSM
RIGHT SHIFT REGISTER
FSM
RIGHT SHIFT REGISTER
32
EC
EQ
EQ
COUNTER
COUNTER
COUNTER
0 TO 7
0 TO 31
0 TO 10416

Instruction Memory

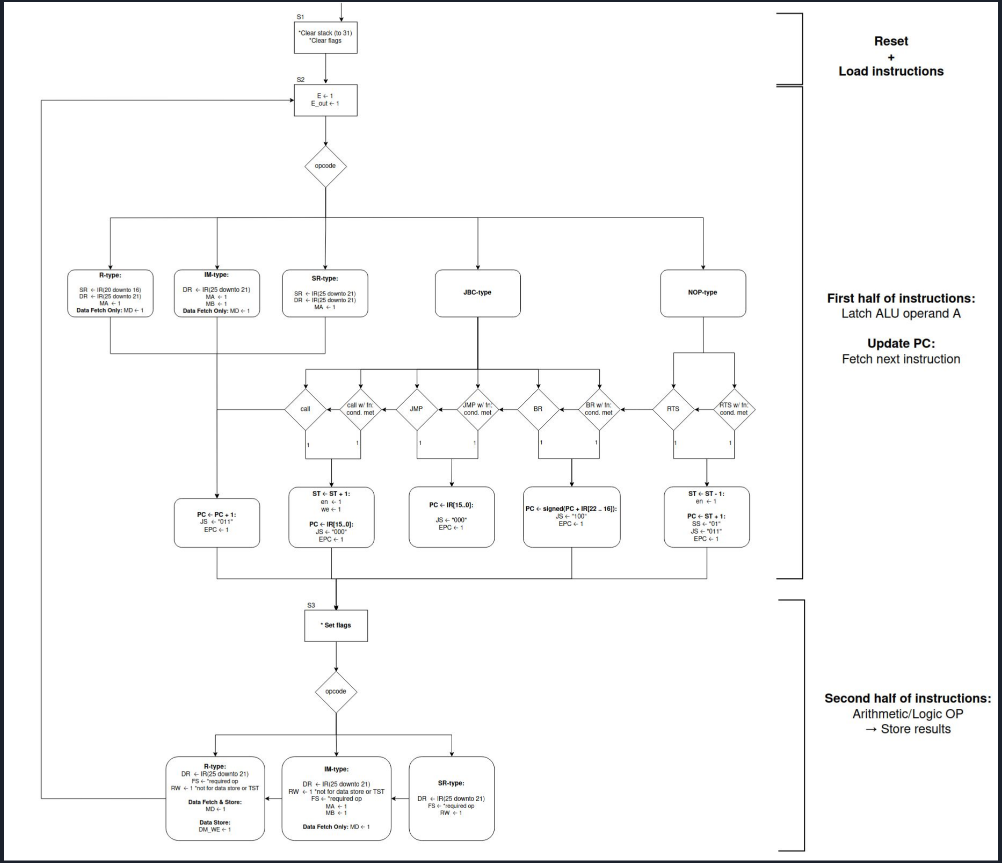
- Single port ram
  - BRAM
  - 1 clock cycle read/write
  - For this design, only 1024 words used
  - Writes occur during load time
  - Generated using Xilinx Block Memory Generator

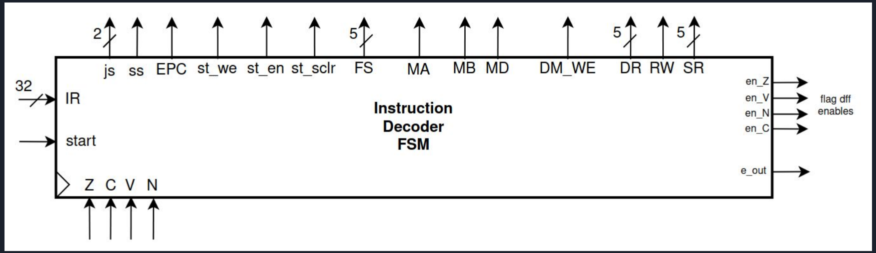
- Control signals
  - Loader: we



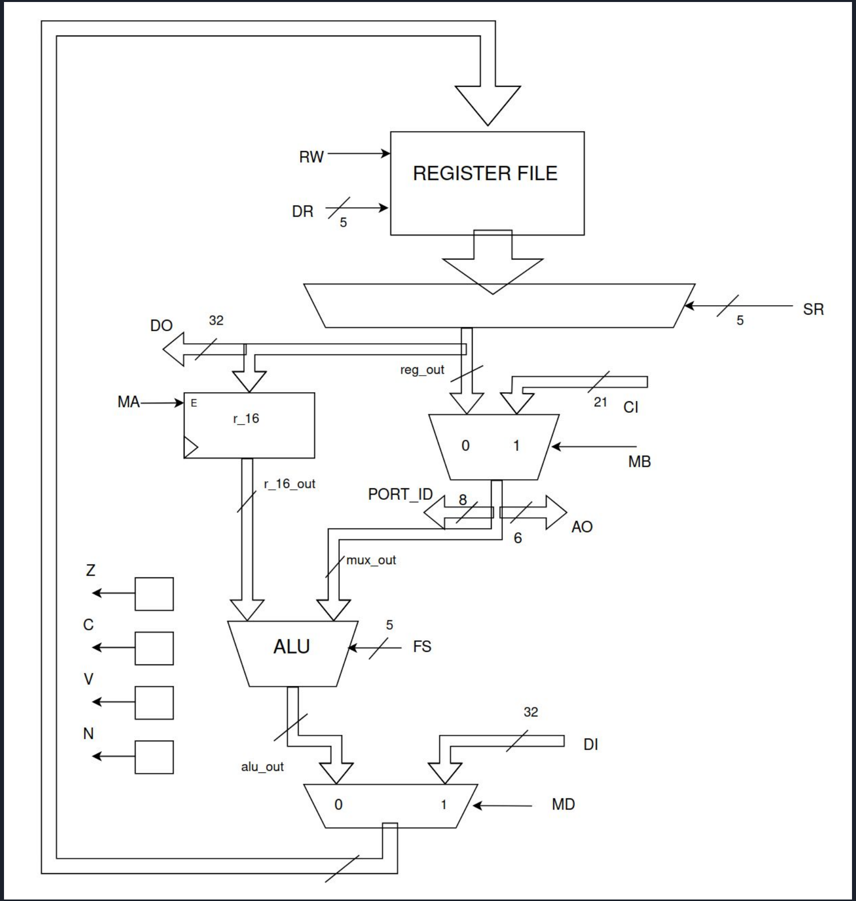
Instruction Decoder

- 3 state FSM
- Fixed length instructions
  - Instruction length: 32 bits
- Each instruction takes 2 clock cycles

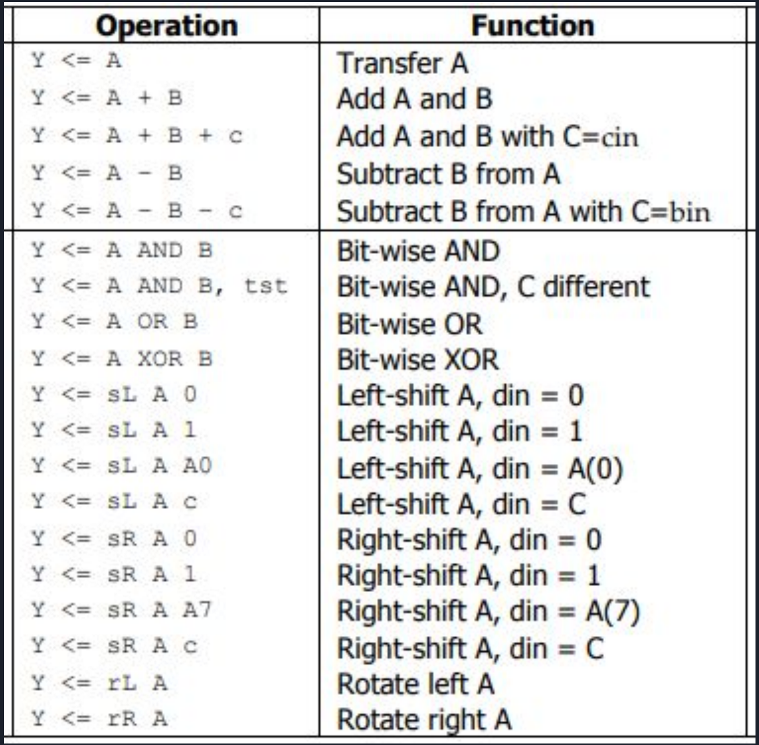




Datapath


REGISTER FILE
RW
DR
5
SR
5
DO
32
reg_out
MA
E
r_16
21
CI
0
1
MB
r_16_out
PORT_ID
8
AO
6
mux_out
Z
C
V
N
ALU
5
FS
32
DI
alu_out
0
1
MD

ALU

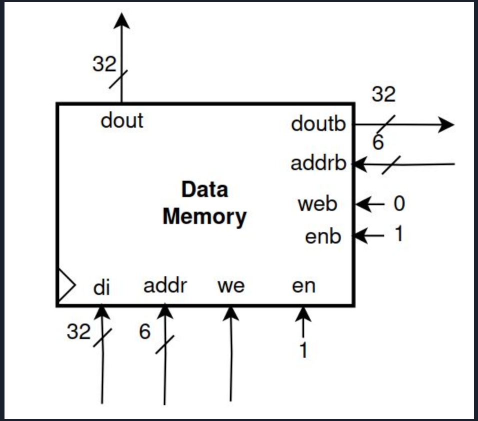


Operation	Function
`Y <= A`	Transfer A
`Y <= A + B`	Add A and B
`Y <= A + B + c`	Add A and B with C=cin
`Y <= A - B`	Subtract B from A
`Y <= A - B - c`	Subtract B from A with C=bin
`Y <= A AND B`	Bit-wise AND
`Y <= A AND B, tst`	Bit-wise AND, C different
`Y <= A OR B`	Bit-wise OR
`Y <= A XOR B`	Bit-wise XOR
`Y <= sL A 0`	Left-shift A, din = 0
`Y <= sL A 1`	Left-shift A, din = 1
`Y <= sL A A0`	Left-shift A, din = A(0)
`Y <= sL A c`	Left-shift A, din = C
`Y <= sR A 0`	Right-shift A, din = 0
`Y <= sR A 1`	Right-shift A, din = 1
`Y <= sR A A7`	Right-shift A, din = A(7)
`Y <= sR A c`	Right-shift A, din = C
`Y <= rL A`	Rotate left A
`Y <= rR A`	Rotate right A

Data Memory

- True dual port RAM
  - BRAM
    - Generated using Xilinx Block Memory Generator
  - 1 clock cycle read/write
  - 64 words
  - Ports
    - A: Datapath
    - B: UART output (read only)

- Control signals
  - ID: we

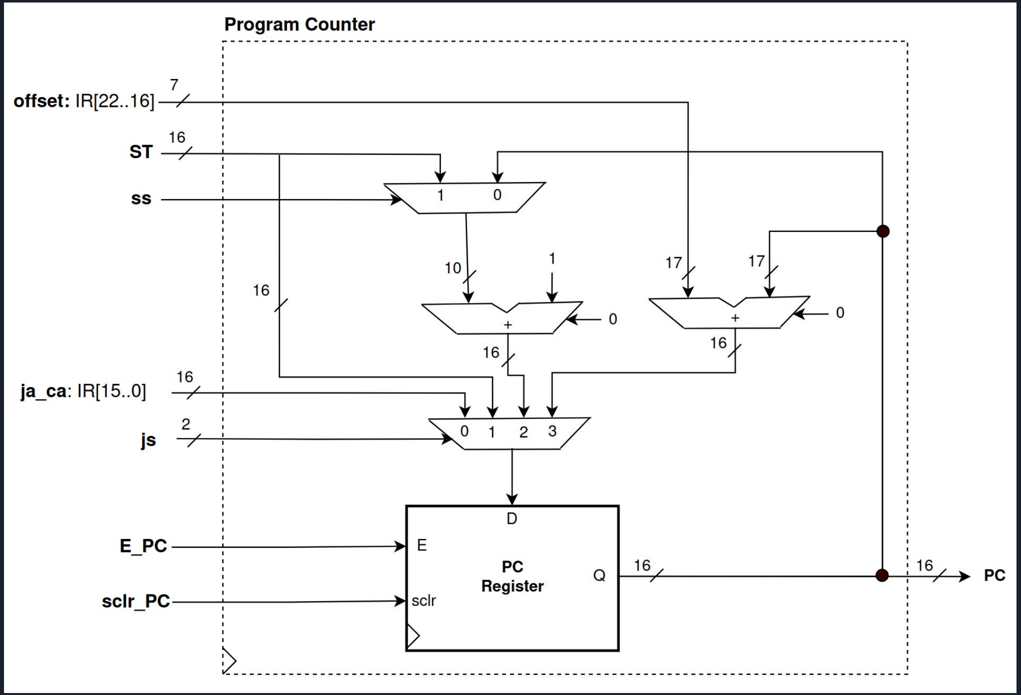


Program Counter

- Operations:
  - Jump & Call
    - Address: IR[15 .. 0]
  - Branch
    - Signed offset: IR[22 .. 16]

- Program counter leads instruction fetch

- Control signals
  - ID: ss, js, E_PC, sclr_PC



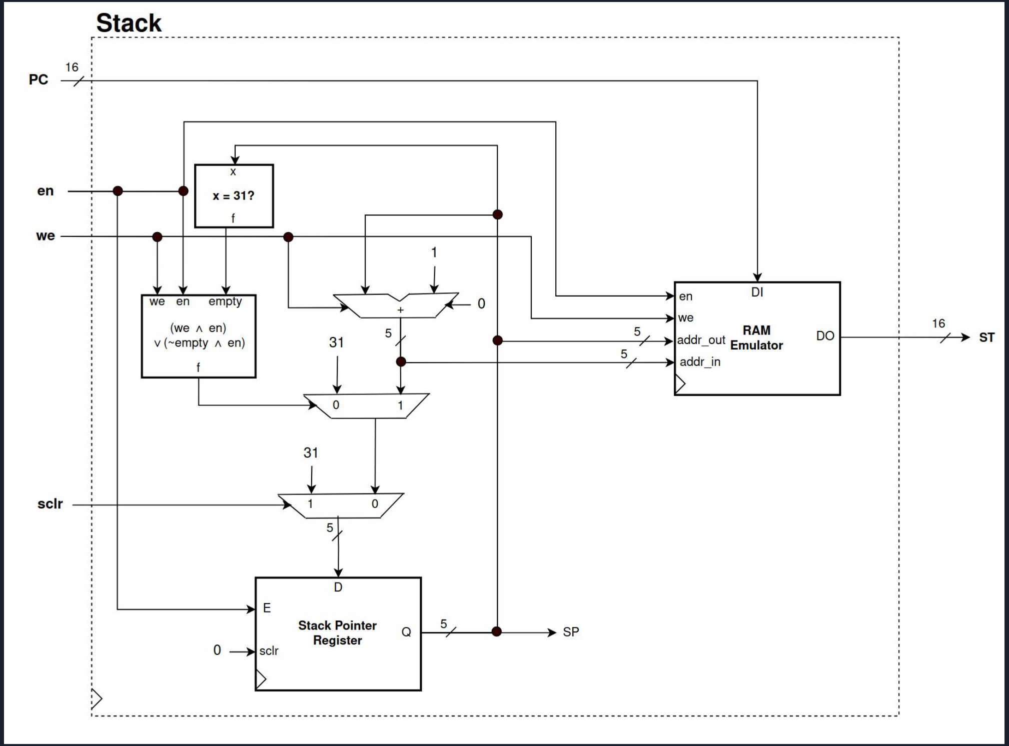
Stack

- 32 Nested Functions
- No read delay
- 1 clock cycle write delay

- Operations:
  - Pop: en ⟵ 1
  - Push: we ⟵ 1, en ⟵ 1
  - Reset: en ⟵ 1, sclr ⟵ 1

- Control signals
  - ID: we, en, sclr

- PC values stored in registers

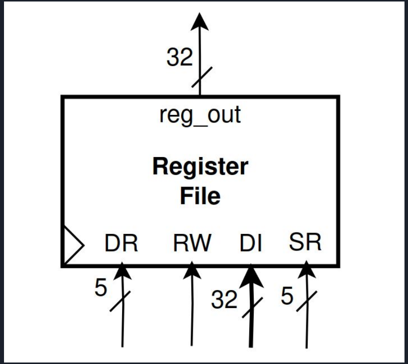


Register File

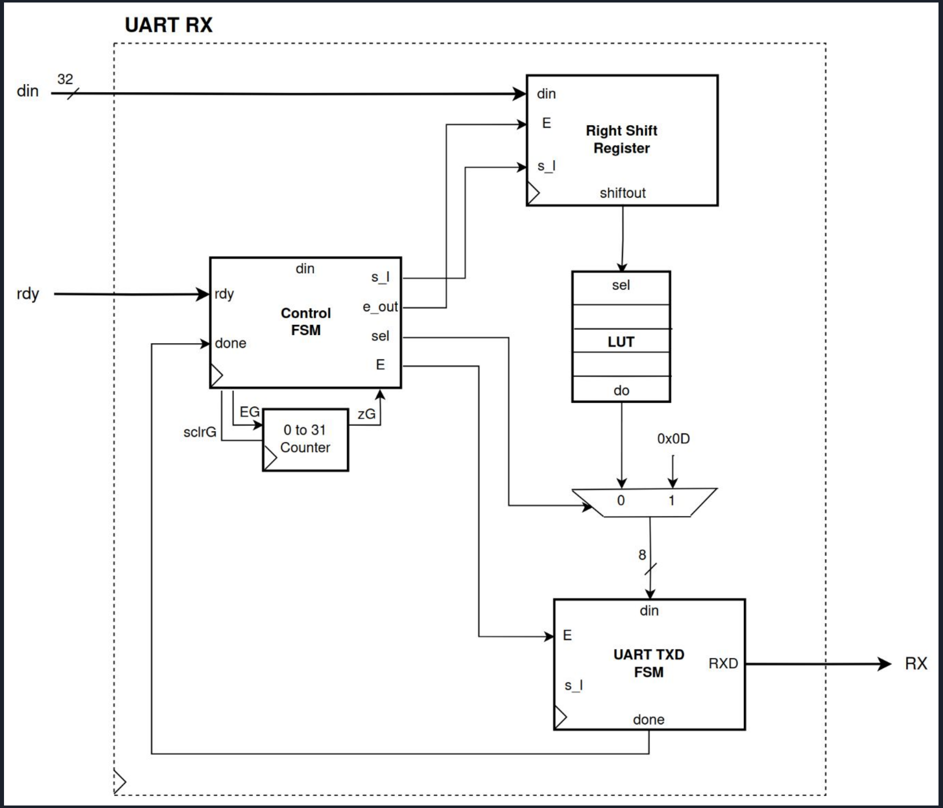
- Contains 32 general purpose registers
- Each register has 32 bit width

- SR - selects source register to put on bus
  - reg_out
- DR - selects which register to write to
  - RW - enables selected register

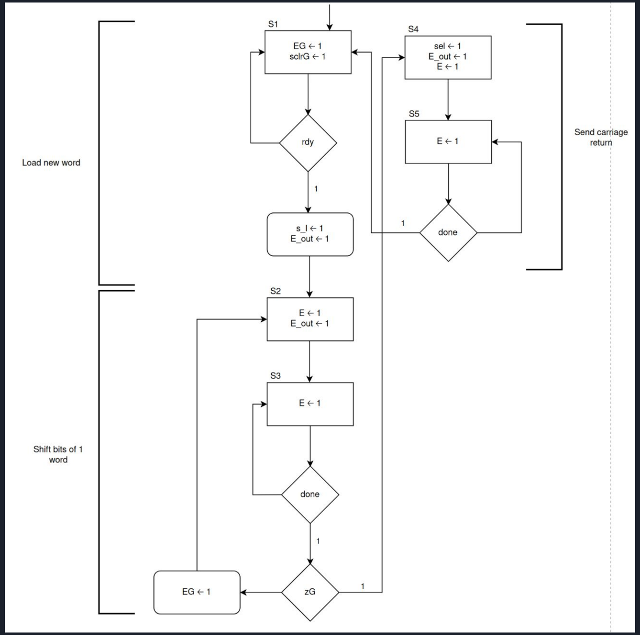
- Control signals
  - ID: DR, SR, RW



Output



UART RX
din
32
rdy
din
E
s_l
Right Shift Register
shiftout
din
rdy
done
s_l
e_out
sel
E
Control FSM
EG
sclrG
0 to 31 Counter
zG
sel
LUT
do
0x0D
0
1
8
din
E
UART TXD FSM
s_l
RXD
done
RX



Load new word
Shift bits of 1 word
Send carriage return
S1
EG ← 1
sclrG ← 1
rdy
1
s_l ← 1
E_out ← 1
S2
E ← 1
E_out ← 1
S3
E ← 1
done
1
zG
1
EG ← 1
S4
sel ← 1
E_out ← 1
E ← 1
S5
E ← 1
done
1

Demo

Test Instructions:

TFR r0, 8: 00000000000000000000000000001000

TFR r1, 4: 00000000001000000000000000000100

TFR r2, 2 : 00000000010000000000000000000010

TFR r3, 1: 00000000011000000000000000000001

STWI r0 63: 10111000000000000000000000111111

STWI r1 62: 10111000001000000000000000111110

STWI r2 61: 10111000010000000000000000111101

STWI r3 60: 10111000011000000000000000111100