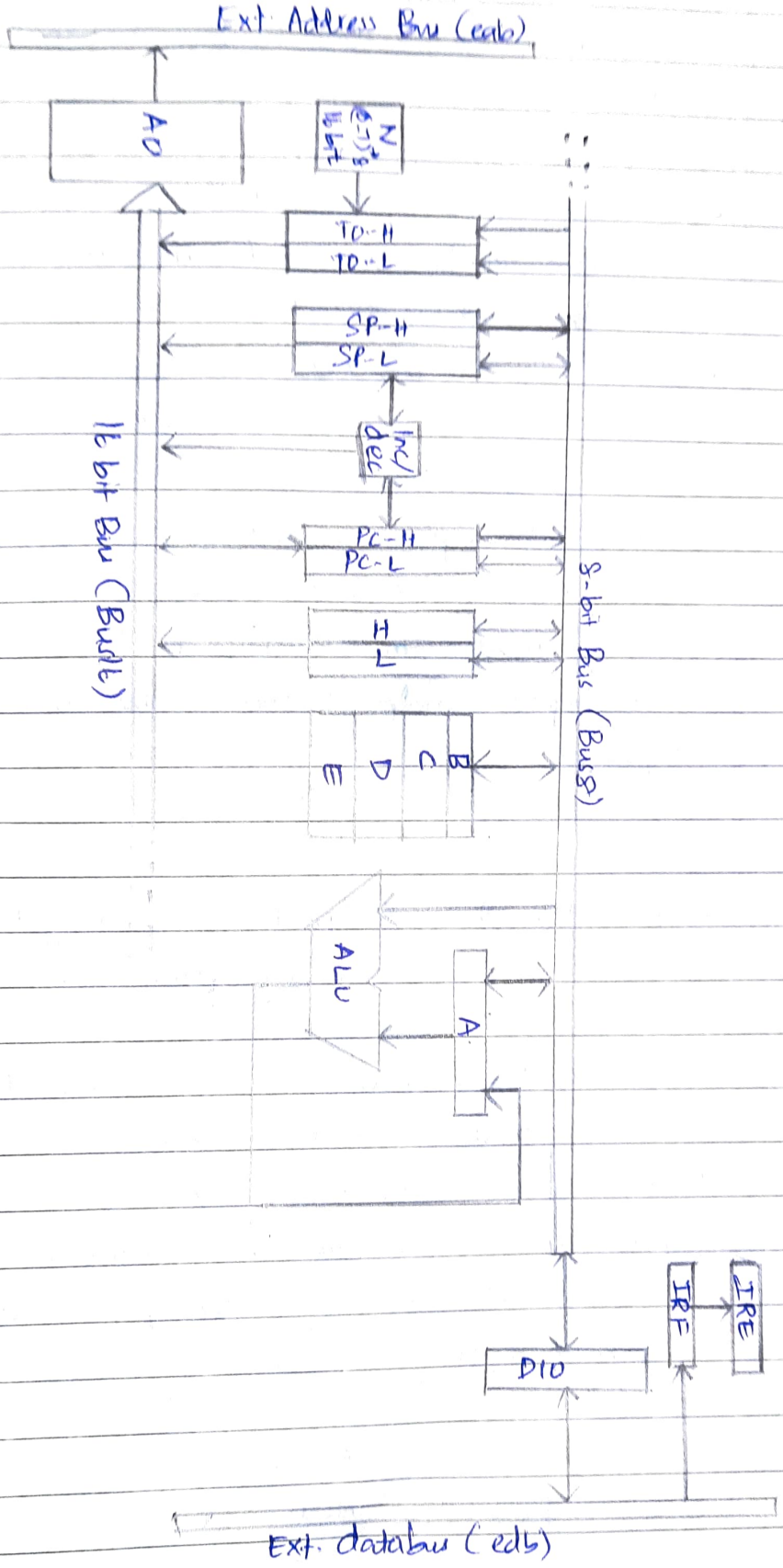
08/04/22

Page No.:
Date:

EE 309 - Assignment 1
Mini - 8085

Gowtham S
20D070031

DATAPATH:



Hardware Flowcharts

Rules of Operation:

- A transfer from source to bus to destination takes one state time.
- A source can drive up to three destination loads.
- When ALU is a destination, the other input is implicitly assumed to be in the Accumulator (A) and the result is also stored in the Accumulator.
- A transfer to the AD buffer activates the on-chip external bus controller. The bus controller postpones the next state until the external transfer is complete.
- Final Assumption is that a register may be incremented or decremented in one cycle itself, if it is not being read from otherwise, in the same cycle. This assumption helps reduce states at certain critical places.

Level 2 Flowcharts (Operation tasks merged with House keeping tasks)

Addressing Mode Sequence – Immediate Addressing:

This addressing mode is common to JMP, CALL, LOAD, STORE. This is the only addressing mode that is worth separating for the execution sequence. Before showing its flowchart, a typical components of a state & a flowchart is given below

	Accu Type
Tasks	ALU Acc
	State ID
	Nxt State

PC → Bus16	MR
→ A0	X-N
PC → INC	AM1
EDB → D10	AM2
D10 → Bus8	R
→ T0-H	X-N
INC → PC	AM2
	AM3
PC → Bus16 → A0	MR
PC → INC	X-N
EDB → D10	AM3
	AM4
D10 → Bus8	R
→ T0-L	X-N
INC → PC	AM4
	SB

Now we can move on to merging the housekeeping tasks with the relevant instructions in an efficient way. JUMP and CALL instructions do not req. housekeeping tasks.

(Note: Instead of having the branch control be connected to the control store, the instruction decoder is made to control the branch control)

Date : / /

Move Instructions:

R → R

	MOVE
RX → Bus8 → RY	R, MR
PC → Bus16 → AD	INC
PC → INC	MVRR1
EDB → IRF	MVRR2
IRF → IRE	R
INC → PC	X-N
	MVRR2
	IB

M → R

	MOVE
HL → Bus16 → AD	MR
EDB → DID	X-N
	MVMR1
	MVMR2
DID → Bus8 → RX	R, MR
PC → Bus16 → AD	INC
PC → INC	MVMR2
EDB → IRF	MVRR2
IRF → IRE	R
INC → PC	X-N
	MVRR2
	IB

R → M

	MOVE
RX → Bus8 → DID	R, MR
PC → INC	INC
PC → Bus16 → AD	MVRM1
EDB → IRF	MVRM2
DID → EDB	MW
HL → Bus16 → AD	X-N
INC → PC	MVRM2
IRF → IRE	IB

IMM → R

	MOVE
PC → Bus16 → A0	MR
PC → INC	INC
EDB → D10	MVIR1
	MVIR2
D10 → Bus8	R
→ RX	X-N
INC → PC	MVIR2
	MVIR3
PC → Bus16 → A0	MR
PC → INC	INC
EDB → IRF	MVIR8
	MVRR2
IRF → IRE	NA
INC → PC	X-N
	MVRR2
	IB

Load/Store Instructions.

Here, the assumption is that the addressing mode sequences have already occured. The following states occur after a sequence branch.

IMM → Rp/Sp

	LOAD
TO-H → Bus8 → RP1	R, MR
PC → INC	INC
PC → Bus16 → A0	LW RP1
EDB → IRF	LWPP2
TO-L → Bus8 → RP2	R
INC → PC	X-N
IRF → IRF	LWRP2
	IB

LOAD

IMM ADDR → R

PC → Bus16 → AD	MR
EDB → DIO	X-N
	LWR1
	MVMR2
DIO → Bus8 → RX	R, MR
PC → INC	INC
PC → Bus16 → AD	MVMR2
EDB → IRF	MVRR2
	NA
INC → PC	X-N
IRF → IRE	MVRR2
	IB

R → IMM ADDR

STORE

RX → Bus8 → DIO	R, MR
PC → INC	INC
PC → Bus16 → AD	SWR1
EDB → IRF	SWR2
T6 → Bus16 → AD	MW
DIO → EDB	X-N
INC → PC	SWR2
IRF → IRE	IB

Arithmetic / Logic Instructions :

R - A

ALU-OP

RX → Bus8 → ALU	
A → ALU	R, MR
PC → INC	OP-F-INC
PC → Bus16 → AD	ALR1
EDB → IRF	MVRR2
ALU → A	
INC → PC	NA
IRF → IRE	X-N
	MVRR2
	IB

IMM-A

	ALU-OP
PC → Busl6 → AD PC → INC EDB → DIO	MR INC ALI1 ALI2
DIO → Buss → ALU A → ALU INC → Pc ALU → A	NA OP-F ALI2 MVIR3
PC → INC PC → Busl6 → Ao EDB → IRF	MR INc MVIR3 MVRR2
INC → PC IRF → IRE	NA X-N MVRR2 IB

Unconditional Jumps:

In foll. instructions, the jump and the CALL instructions req. the housekeeping tasks to be performed first.

	JUMP
TO → Busl6 → PC, AO EDB → IRF PC → INC	R, MR INC JMP 1 MVRR2
INC → PC IRF → IRE	NA X-N MVRR2 IB

CALL

PC-H → Bus8 → DIO SP → Bus16 → AD SP → INC	MW INC CLU1 CLU2
PC-L → Bus8 → DIO INC → SP INC → Bus16 → AD	MW X-N CLU2 CLU3
TD → Bus16 → PC SP → INC INC → SP	NA INC CLU3 CLU4
PC → INC PC → Bus16 → AD EDB → IRF	MR INC CLU4 MVRR2
INC → PC IRF → IRE	NA X-N MVRR2 FB

RET

SP → DEC DEC → SP DEC → Bus16 → AD EDB → DIO	MR DEC RET1 RET2
SP → DEC DIO → Bus8 → PC-L	NA DEC RET2 RET3
DEC → Bus16 → AD DEC → SP EDB → DIO	MR X-N RET3 RET4
DIO → Bus8 → PC-H	NA X-N RET4 CLU4
PC → INC PC → Bus16 → AD EDB → IRF	MR INC CLU4 MVRR2
INC → PC IRF → IRE	NA X-N MVRR2 FB

Conditional Jumps:

Here again, the Jump and the Call instruction require the house keeping tasks to be performed first

CARRY	JUMP
PC → Bus lb → AD PC → INC EDB → IRP	MR INC JMPC BC

INC → PC EDB → IRE	NA X-N MVRR2 IB	TD → Bus lb → PC, AD EDB → IRF PC → INC	R, MR INC JMP1 MVRR2
		INC → PC IRF → IRE	NA X-N MVRR2 IB

ZERO	CALL
PC → Bus lb → AD PC → INC EDB → IRF	MR INC JMPC BC

0 1

INC → PC EDB → IRE	NA X-N MVRR2 IB	PC-H → Bus 8 → DIO SP → Bus lb → AD SP → INC	MW INC CLV1 CLV2

↓ Rest of the States follow

PC → Bus lb → AD PC → INC EDB → IRF	MR INC JMPC BC

0 1

INC → PC EDB → IRE	NA X-N MVRR2 IB	SP → PEC DEC → SP PEC → Bus lb → AD EDB → DIO	MR DEC RET1 RET2

↓ Rest of the States follow

With this we are done with all the flowcharts which have been optimized to minimise the total number of states. Next we list out all the states and their corresponding addresses in the control store:

4-3 \ 2-0	000	001	010	011	100	101	110	111
00	AM1	AM2	AM3	AM4	MVRR1	MVRR2	MVMR1	MVMR2
01	MVRM1	MVRM2	MVIR1	MVIR2	MVIR3	LWRP1	LWRP2	LWR1
10	SWR1	SWR2	ALR1	ALI1	ALI2	JMP1	CLU1	CLU2
11	CLU3	CLU4	RET1	RET2	RET3	RET4	X	JMP2

Control Store:

Now we decide the control bits that the datapath requires.

Control bits and decode logic:

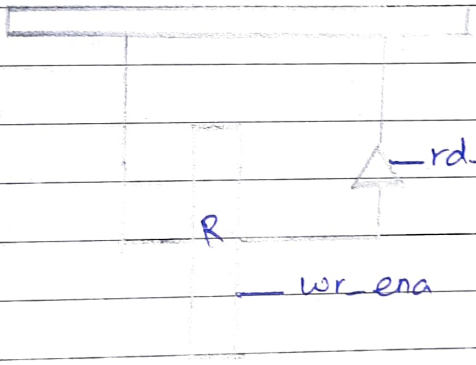
Registers:

The connection of the registers to the Bus8 is what we are commenting on here. We can see from the adjacent image that every register will need 2 bits to control it. But we can't keep copies of a state so that it can operate on from the instruction decoder. Hence from the instruction decoder we have two addresses for the two registers each instruction may need to read or modify.

In the control store hence we have 3 bits, because all possibilities are never possible. The representation format is OP(rd_ena wr_ena)

Bit Combinations	Register 1	Register 2
0xx	NULL (00)	NULL (00)
100	Read (10)	NULL (00)
101	Read (10)	Write (01)
110	NULL (00)	Write (01)
111	Write (01)	NULL (00)

These bits refer to registers A, B, C, D, E, H, L, SP-H & SP-L and because they are in total 9 registers we have a 4-bit address coming from the Instruction decoder which passes through a logic decoder and the above outputs are ANDed with ~~the~~ decoder output ~~to~~ give the signal to the individual registers ~~The logic for A~~.



ALU:

Essentially the ALU needs to perform 4 operations and because the ALU is Combinational we don't as such need anything from the control word. The instruction that must be performed will directly originate from the instruction decoder, and it will always produce the output. But the flags will be changed and the result will be stored in the accumulator only when the accumulator write is activated ~~for~~ to read from the ALU. The control for the setting of flags will be discussed along with the elucidation of ~~ALU~~ Accumulator.

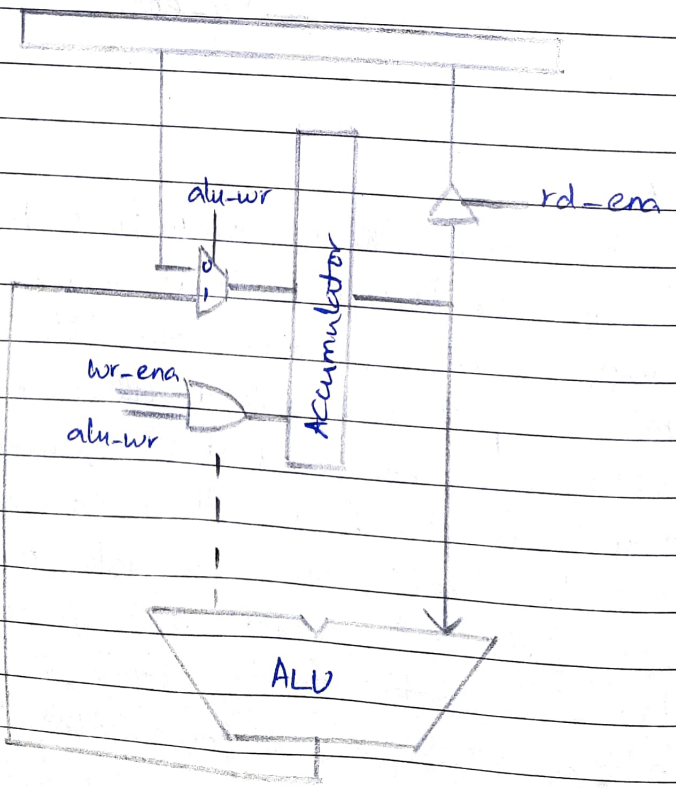
Page No. :
Date : / /

Accumulator:

Because the accumulator can be written both from the bus and through the ALU, we add another signal which need not be decoded and hence is represented by a bit in the control word itself. This bit is called the alu_wr.

Also to elaborate on the ALU control, there must be two bits in the instruction to decide which instruction the ALU must perform. So we directly connect these to the ALU as control. Also the 2 operations which use flags are ADD and SUB but both the instructions always use the Carry flag, so no control is required for that.

While the other instructions set all flags, the AND operation will not set the zero flag, So we can enable the condition code register with "alu_wr OR ANA" where ANA represents the code for AND operation that is sent to the ALU.

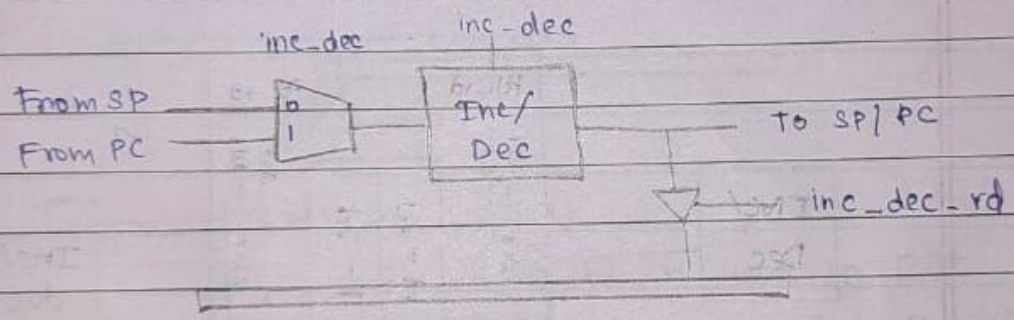


PAGE:
DATE: / /

INC/DEC

Assuming that we have only one block which performs both functions but only one at a time we have the following. We need two bits to control this piece of hardware, the inc_dec which controls what the block must do and the inc_dec_rd which allows it to write to the bus. The output of this block is also connected at the input of the SP and PC through muxes which we will see in a short while.

Bit Pattern	inc_dec	inc_dec_rd
ab	a	b



Program Counter

The program counter consists of two blocks. Each of it can read and write to SBUS. Also as a 16bit register it can read from the incrementer, the 16BUS and also write to the 16BUS. In total the control signals required for the program counter is 6 as described below

In the figure there are 7 inputs but the enables of the register can be written in terms in terms of sel1 and sel2 as follows.

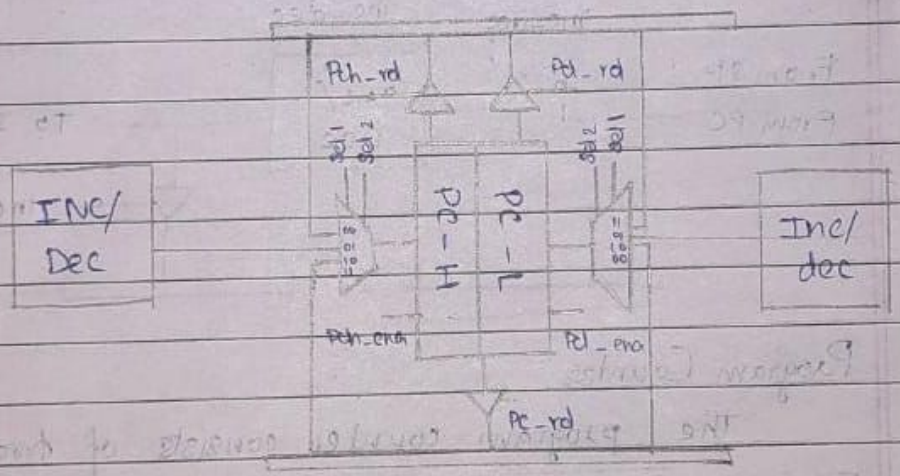
pchena $\leftarrow \overline{sel1.sel2}.ena$ and

pdena $\leftarrow (sel1 + sel2).ena$

PAGE:
DATE: / /

This is possible because of specific arrangements of the multiplexers and their inputs. For encoding this in control word we need 3 bits as follows

Bit Pattern	sel1	sel2	Pch_rd	Pcl_rd	PC_rd	ena
rab	a	b	0	0	0	1
000	0	0	0	0	0	0
001	0	0	0	0	1	0
010	0	0	0	1	0	0
011	0	0	1	0	0	0



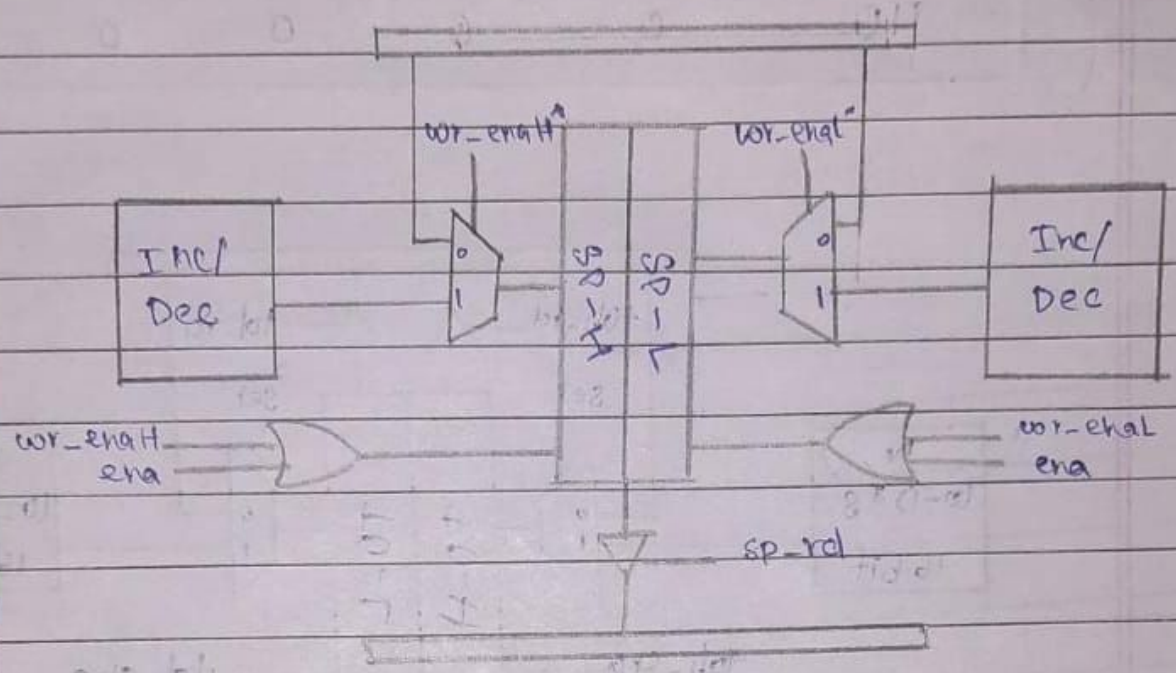
Stack Poinder

We note that the stack pointer can only write to the 16BUS and not read from it. It can only read from the 8BUS and the INC/DEC block. Also as it was involved with the registers, it already had two enables for the individual 8 bit registers. So we add two multiplexers, another write enable signal, and the read enable signal.

Here the wr_ena and wr_enal refer to those which are decoder from the instruction and the register part of the control word.

PAGE:
DATE: / /

we have hence two bits in the control word for ena and sp-rd, because they can be separately enabled and disabled.



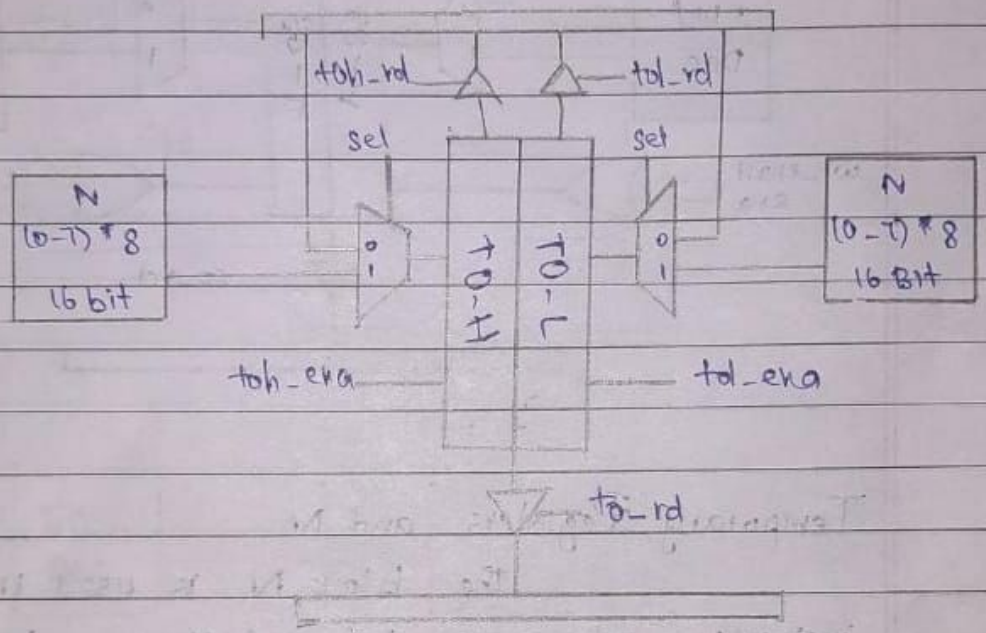
Temporary Registers and N

The block N is used in RST instruction where the value of n, is directly extracted from instruction decoder. Hence no special control signals are required by it.

Now for the temporary registers, we need a multiplexor to choose b/w the block N and the 8 BUS and hence the relevant enable signals. Also we need a read enable bit to allow it to write to the 16 BUS. All of this can be coded in 3 bits of the control word.

PAGE:
DATE: / /

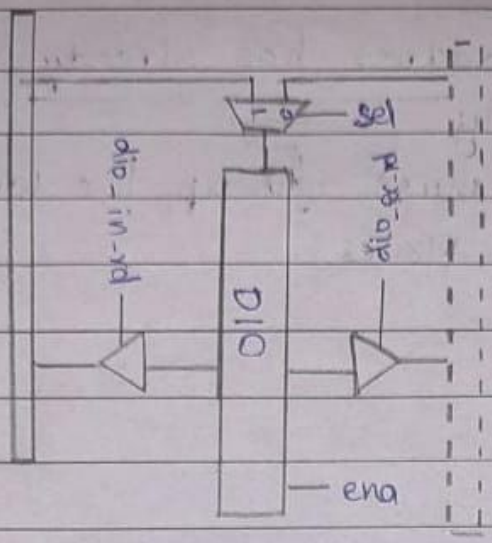
Bit Pattern	to_rd	toh_ena	tol_ena	sel	toh_rd	tol_rd
abc	a	b	c	b.c	a	a
100	1	0	0	0	0	0
101	0	0	0	0	0	1
110	0	0	0	0	1	0



Data Input / Output (DIO)

This register has outputs at two buses and two inputs as well. Hence we need 4 control signals which can be encoded into 3 bits of the control store.

Bit pattern	dio_in_rd	dio_ex_rd	ena	Sel
0xx	0	0	0	x
100	0	0	1	0
101	0	1	0	x
110	1	0	0	x
111	0	0	1	1



with this we are left with only 3 registers namely AD, IRF and IRE. But we note that these three need only one signal each because their outputs are always enabled and have only one input each. Hence we only need to control each register's enable.

This leads us finally to the control word structures as follows.

Control Word

26								7 6		5 4	0
Regs	ACC	INC/DEC	PC	SP	IO	DIO	AD	IRF	IRE	Type	Address / Next

The type field tells what kind of branch must be performed next

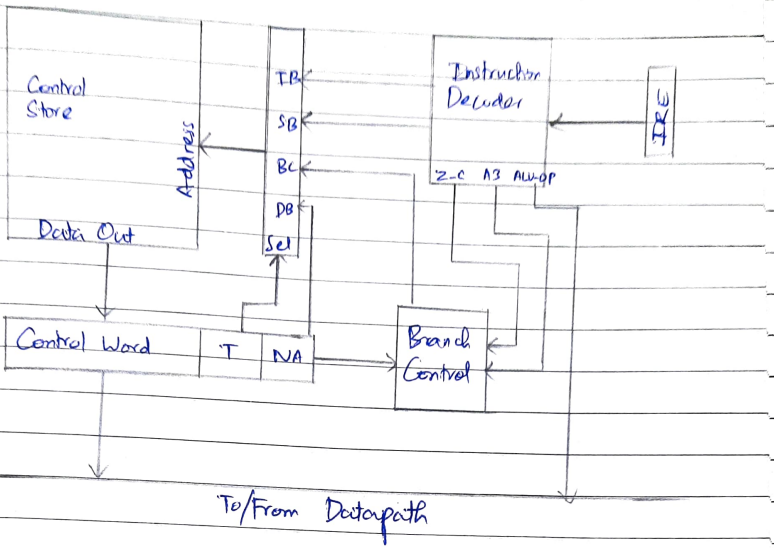
Bit Pattern	Branch Type
00	SB
01	IB
10	BC
11	DB

The next address has already been written out previously.
Now we must fill up the control store for all 32 states.

The Control Word

States	Registers			A	INC/ DEC		PC			SP		T0			DIO			AO	IRF	IRE	Type		Next Address				
AM1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1
AM2	0	0	0	0	0	0	1	0	1	0	0	0	1	0	1	1	0	0	0	0	1	1	0	0	0	1	0
AM3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	1
AM4	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	X	X	X	X	X
MVRR1	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1
MVRR2	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	1	X	X	X	X	X
MVMR1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	1	1	1
MVMR2	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	1	0	1	1	0	0	1	0	1
MVRM1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	1	1	0	1	1	0	1	0	0	1
MVRM2	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	1	0	1	0	1	X	X	X	X	X
MVIR1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	0	1	1
MVIR2	1	1	1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	1	1	0	1	1	0	0
MVIR3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1
LWRP1	1	1	1	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	0	1	1	0	1	1	1	0
LWRP2	1	1	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	X	X	X	X	X
LWR1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1	0	0	1	1	1
SWR1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	1	0	0	0	1
SWR2	0	0	0	0	0	0	1	0	1	0	0	1	0	0	1	0	1	1	0	1	0	1	X	X	X	X	X
ALR1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1
ALI1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	1	1	0	1	0	0
ALI2	0	0	0	1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	0	0	1	1	0	1	1	0	0
JMP1	0	0	0	0	0	0	1	1	0	0	0	1	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1
CLU1	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1	1	1	0	0	1	1	1	0	1	1	1
CLU2	0	0	0	0	0	1	0	1	0	1	0	0	0	0	1	1	1	1	0	0	1	1	1	1	0	0	0
CLU3	0	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1
CLU4	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1
RET1	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1	1	1	0	1	1
RET2	0	0	0	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	1	0	0
RET3	0	0	0	0	1	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1	1	1	1	0	1
RET4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1
RST1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	1	1	0	1	1	0
JMPC	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	1

Controller Organisation:



Branch Control.

The branch control is mainly controlled by the instruction decoder itself. The signal z-c tells the branch control as to which flag to branch based on. The addresses to be jumped to is also supplied by the instruction decoder.

The default location to be branched to if the concerned flag is zero is taken directly from the prev. control word. But if the condition evaluates to true, the branch control provides the address as it gets it from A3 from the instruction decoder. This occurs for three instructions and hence the instruction decoder can be hardcoded with the branch addresses and the concerned flag using a look-up table kind of structure. This allows me to have a generic branch state which is used by all the three branch instructions.