

EN2031 - Fundamentals of Computer Organization and Design



ISA and Microarchitecture Design

Prepared by:

DULNATH W.H.R :210152E

U.M.Y.B. ALAHAKOON :210027C

SHAMIKA K.A.M. :210600D

a.]

i.) List of Required Instructions

Instruction	OPCODE	Operation	Meaning
ADD R1, R2	00000	$R1 = R1 + R2$	Perform the addition on 2 registers
SUB R1, R2	00001	$R1 = R1 - R2$	Performs subtraction on 2 registers
OR R1, R2	00010	$R1 = R1 \mid R2$	Performs bitwise OR operation on 2 registers
AND R1, R2	00011	$R1 = R1 \& R2$	Performs bitwise AND operation on 2 registers
PUSH Rd	00100	$R7 = R7 - 1$ $DM[R7] = R1$	Pushes a value onto the stack
POP Rd	00101	$R1 = DM[SP]$ $R1 = DM[R7]$	Take off the stack's top element and store it to R1
LD R1, R2	00110	$R1 = DM[R2]$	Load the data in R2 place to R1 register
ST Rd, R1	00111	$DM[R1] = Rd$	Store a value from a register to Data Memory
MOV R1, R2	01000	$R2 = R1$	Moves a value from one register to another
BEQ R1, IMO	01001	If SREG [0] = 1 $PC = PC + IMO$	Branch if equal.
BLE IMO	01010	If SREG [1] = 1 $PC = PC + IMO$	Branch if less than or equal.
BGE IMO	01011	If SREG [2] = 1, $PC = PC + IMO$	Branch if greater than or equal
BR IMO	01100	$PC = PC + IMO$	Unconditional branch instruction
LDI8 Rd, IMO	01101	$Rd = IMO$; Sign-extended to 16 bits	Load immediate 8-bit constant
JMP R1, IMO	01110	$R1 = PC$ $PC = PC + IMO$	Put the PC that is currently in R1 and increase it by IMO
LOADA R1, R2	01111	$R1 = DM[R2]$ $R2 = R2 + 1$	Load the data in R2 register place to R1 register and increment R2 by one

Rd = Memory Read

R0-R6 = Other General-Purpose Registers

R7= Stack Register

IMO= Immediate Operand





PC= Program Counter Value

ii.)

RD: Destination Register

RS: Source Register

Instructions for the ALU:

-  ADD
-  SUB
-  AND
-  OR

OPCODE 5 Bits	RD (3 Bits)	RS (3Bits)	
---------------	-------------	------------	--

Instructions for the Control Flow:

-  BR
-  BEQ
-  BLE
-  BGE
-  JMP

OPCODE (5Bits)	RD/RS (3 Bits)	Signed Immediate Operand
----------------	----------------	--------------------------

Instructions for the Loading:

-  LD
-  LDI8

OPCODE (5Bits)	RD (3Bits)	RS (3Bits)	Immediate Operand
----------------	------------	------------	-------------------

Instructions for the Move:

-  MOV

OPCODE (5Bits)	RD (3Bits)	RS (3Bits)	
----------------	------------	------------	--

POP Operation:

OPCODE 5(Bits)	RD/RS (3Bits)	
----------------	---------------	--

PUSH Operation:

OPCODE (5Bits)	RS (3Bits)		
----------------	------------	--	--

Explanations:

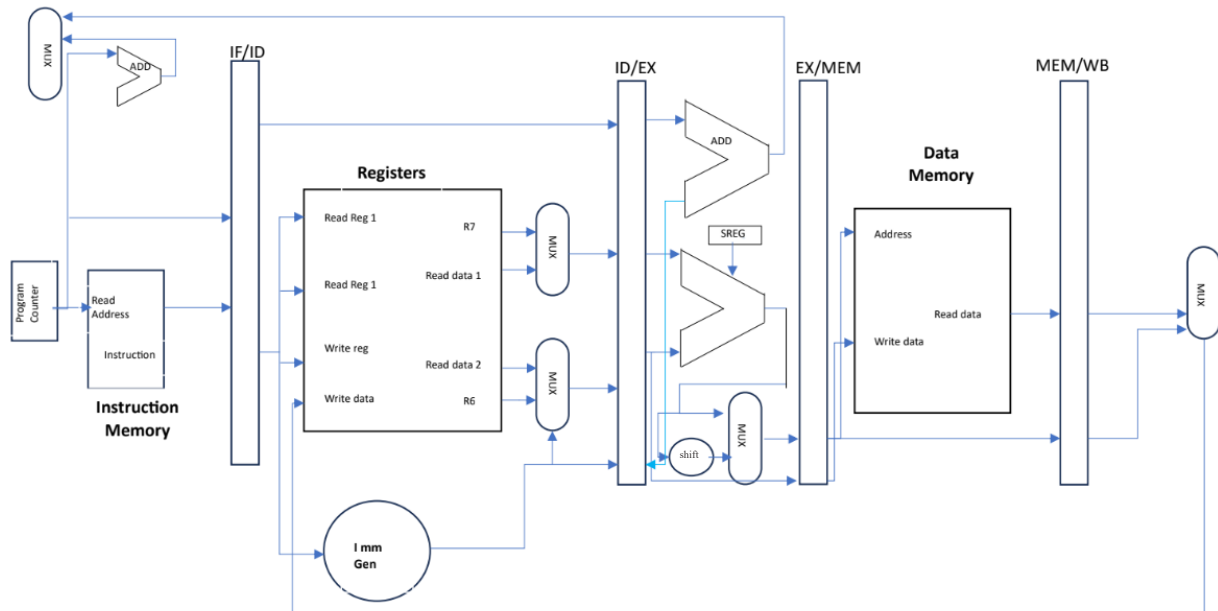
To maximize consistency, efficiency, and clarity, this instruction set design makes a few format options. ALU instructions use a standard 3-operand and 5-bit OPCODE format for clear and efficient arithmetic operations. Control flow instructions employ relative immediate values for streamlined branching. They use 5-bit OPCODE and 3bit Register (Either RS or RD) with signed immediate operand. Control flow instructions simplify branching by using relative immediate values. LOAD instructions use 5-bit OPCODE and 3 Operands. Load instructions allow for flexible data transmission by including source and destination registers. In the POP Operation OPCODE takes 5 bits and operand takes 3 bits and other bits are extras. Same procedure uses in the PUSH Operation. These decisions ensure that the instruction set is well-suited for high-speed, Industry 4.0 compliant factory operations, offering efficiency and clarity in complex computational tasks.

B).

For opcodes, we may utilize a simple binary encoding that allows for up to 16 distinct opcodes by using 5 bits for instruction selection. We use 5 bits because decoding is easier when we have an extra bit in the opcode.

Instruction	OPCODE
ADD	00000
SUB	00001
OR	00010
AND	00011
PUSH	00100
POP	00101
LD	00110
ST	00111
MOV	01000
BEQ	01001
BLE	01010
BGE	01011
BR	01100
LDI8	01101
JMP	01110
LOADA	01111

C).

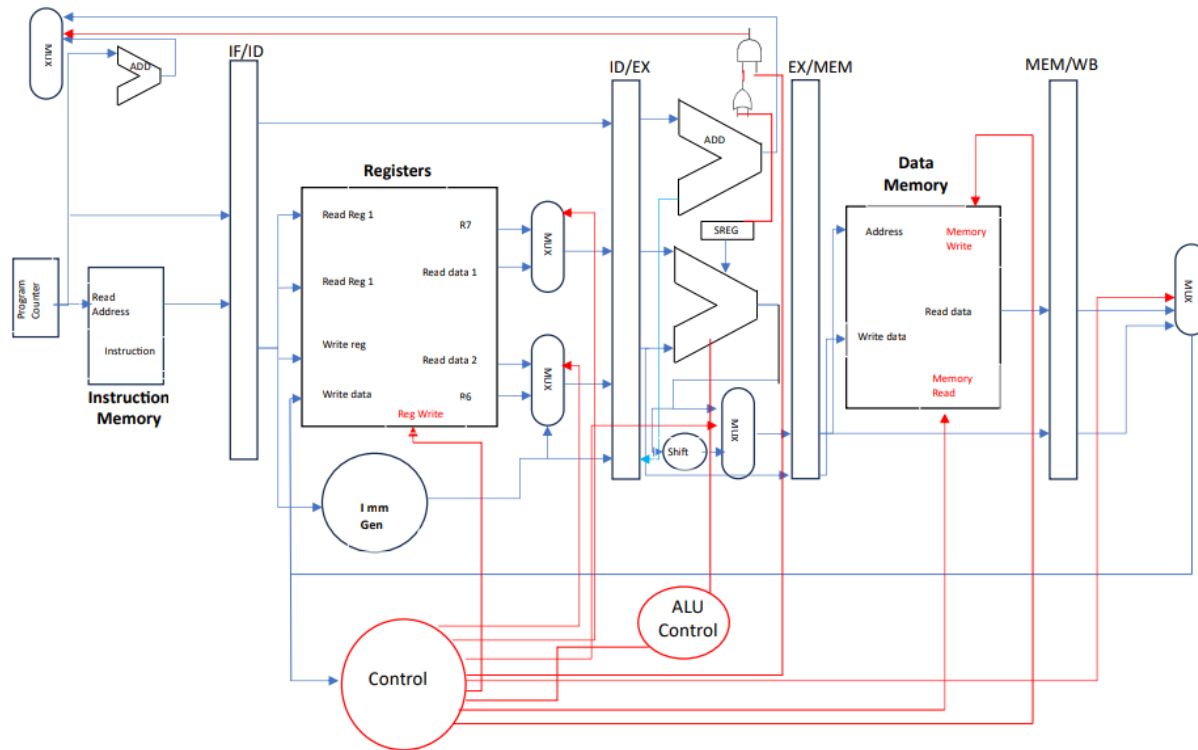


D).

i.) Hardwired Micro Architecture

- The use of a hardwired microarchitecture design approach is driven by the need for speed, predictability, simplicity, and customization in the custom processor, aligning with the requirements, because control signals are generated by the hardware. Since there are only small number of instructions, it is better to have this architecture rather than using microprogramming. Also, it is faster.

ii.)



Based on the type of the instruction the controller is able to control the components in the microarchitecture. We use 5 bits opcode although we have 15 instructions (4-bit OPCODE enough) easy to decode the instructions.

Control Signal	Memory Read	Memory to Reg	ALU Operation	Memory Write	ALU SRC	Reg Data Write	Reg Address Write	Branch	Address ALU OP
Instruction									
ADD	0	0	000	0	00	1	0	000	0
SUB	0	0	001	0	00	1	0	000	0
OR	0	0	010	0	00	1	0	000	0
AND	0	0	011	0	00	1	0	000	0
MOV	0	0	100	0	00	1	0	000	0
LOAD	1	1	000	0	00	1	0	000	0
PLOAD	1	1	000	0	00	1	1	000	1
POP	1	1	000	0	00	1	1	000	0
STORE	0	0	000	1	00	0	0	000	0
UILD	0	0	010	0	01	1	0	000	0
JMP	0	0	000	0	00	0	0	100	0
BEQ	0	0	001	0	10	0	0	101	0
BLT	0	0	001	0	10	0	0	110	0
BGT	0	0	001	0	10	0	0	111	0

ALU control signals

000	ADD
001	SUB
010	OR
011	AND

Branch Control

000	Don't branch
100	Unconditional
101	Jump if equal
110	Jump if less than
111	Jump if greater than

Memory to Register (What is written to the write register)

0	ALU output
1	Data read from memory

Address ALU Operation

0	Ra - 1
1	Ra +1
