Computer Organization and Architecture Laboratory

Group 20

Sourodeep Datta - 21CS10064

Mihir Mallick - 21CS30031

1. Instruction format and Encoding

1.1 R type instructions:

Opcode	Source 1	Source 2	Destination	NA	Function code
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Instruction	Usage	Opcode	Funct code
ADD	ADD rd, rs, rt	000000	100000
SUB	SUB rd, rs, rt	000000	100010
AND	AND rd, rs, rt	000000	100100
OR	OR rd, rs, rt	000000	100101
XOR	XOR rd, rs, rt	000000	100110
NOT	NOT rd, rs, rt	000000	100111
SLA	SLL rs, rt	000000	000000
SRL	SRL rs, rt	000000	000010
SRA	SRA rs, rt	000000	000011

1.2 I type instructions:

Opcode	Source	Destination	Immediate Value
6 bits	5 bits	5 bits	16 bits

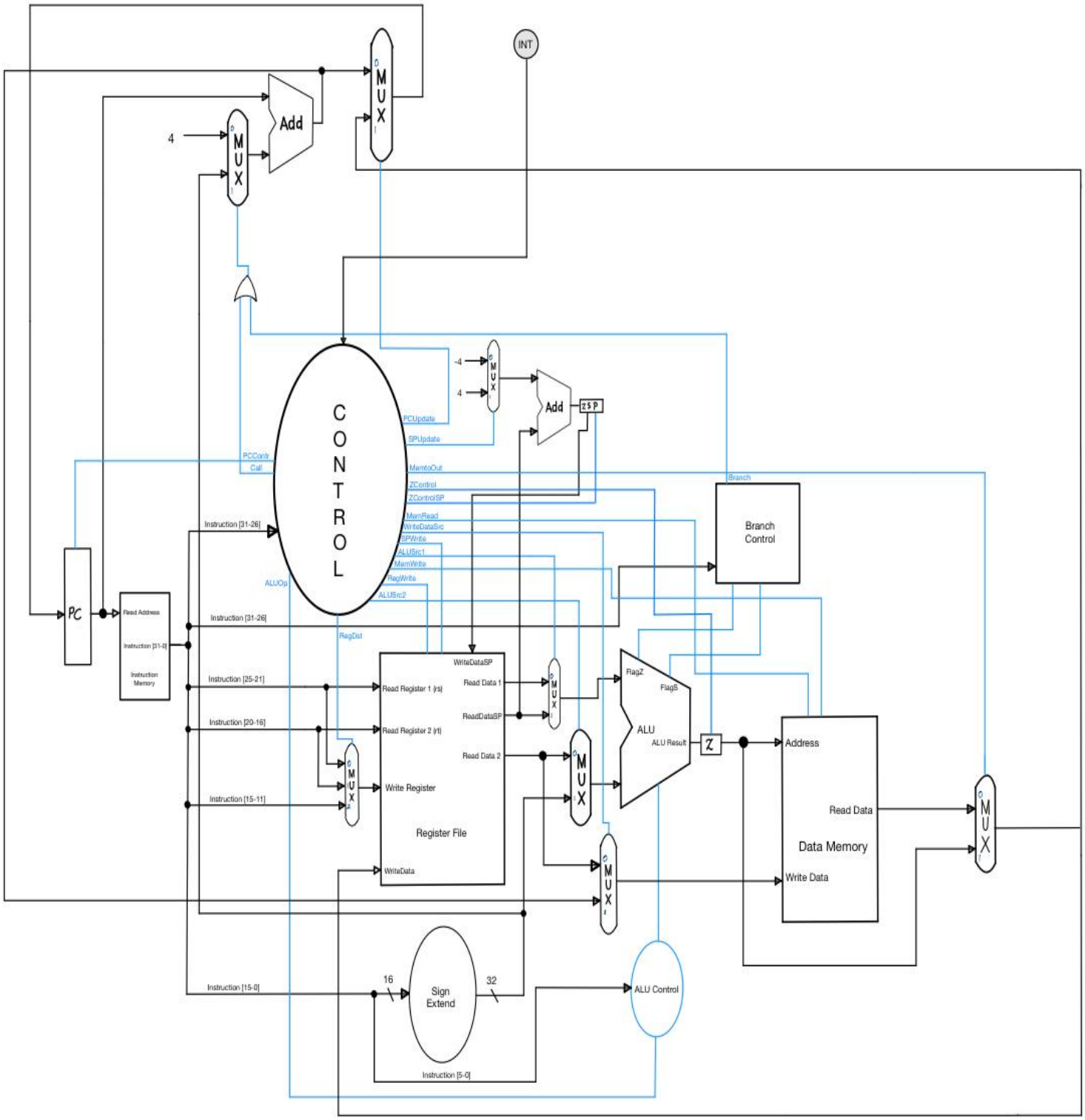
Instruction	Usage	Opcode
ADDI	ADDI rs, imm	000001
SUBI	SUBI rs, imm	000010
ANDI	ANDI rs, imm	000011
ORI	ORI rs, imm	000100
XORI	XORI rs, imm	000101
NOTI	NOTI rs, imm	000110
SLLI	SLAI rs, imm	000111
SRLI	SRLI rs, imm	001000
SRAI	SRAI rs, imm	001001
LD	LD rt, rs, imm	001010
ST	ST rt, rs, imm	001011
LDSP	LDSP rt, rs, imm	001100
STSP	STSP rt, rs, imm	001101
BR	BR imm	001110
BMI	BMI rs, imm	001111
BPL	BPL rs, imm	010000
BZ	BZ rs, imm	010001
PUSH	PUSH rt	010010
POP	POP rt	010011

CALL	CALL imm	010100
RET	RET	010101
MOVE	MOVE rt, rs	010110
HALT	HALT	100000
NOP	NOP	100001

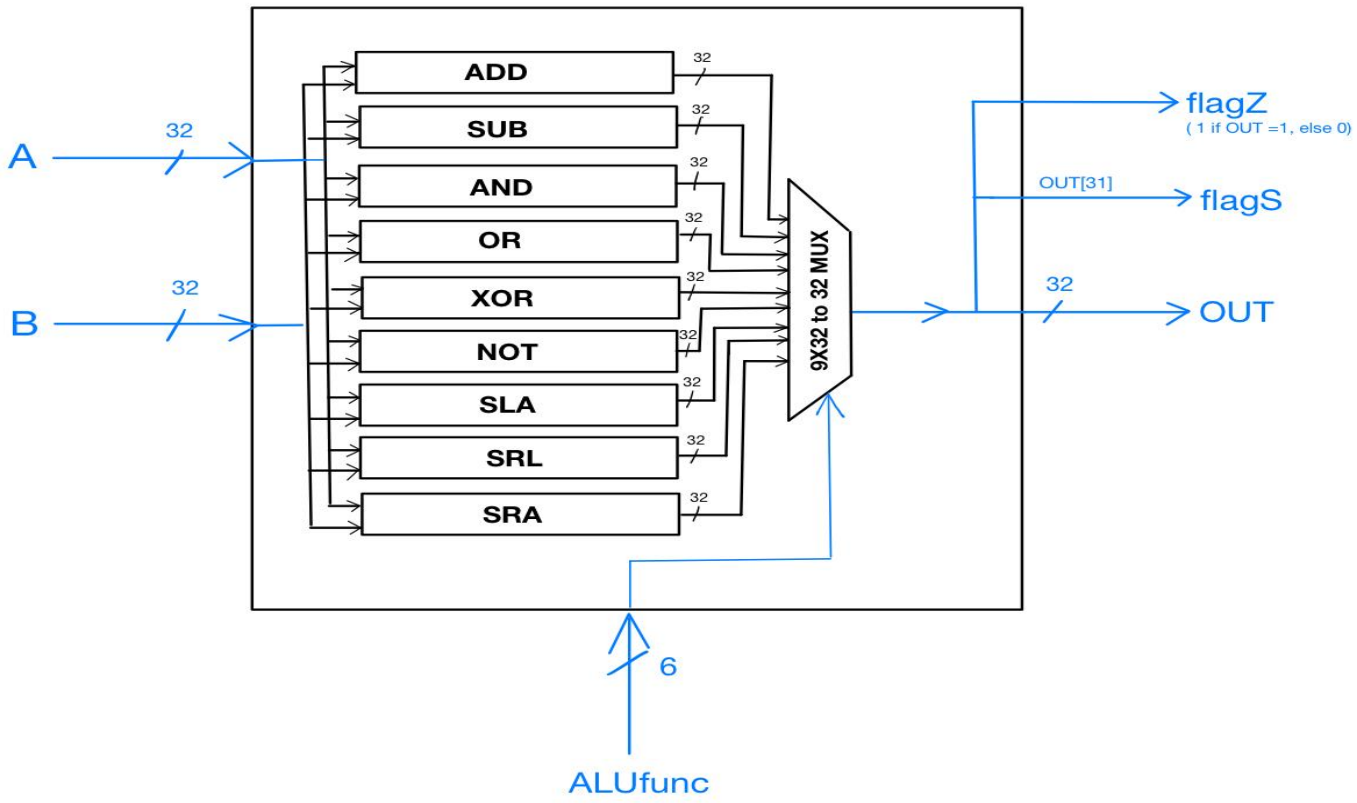
2. Register Usage Conventions

Register Number	Name	Purpose
0	R0	Hardwired to 0
1-15	R1-R15	General Purpose
16	R16	Stack Pointer

3. Datapath


INT
MUX
Add
4
CONTROL
PCUpdate
SPUpdate
MemtoOut
ZControl
ZControlSP
MemRead
WriteDataSrc
SPWrite
ALUSrc1
MemWrite
RegWrite
ALUSrc2
ALUOp
RegDst
PCCont
Call
Branch
Branch Control
PC
Read Address
Instruction [31-0]
Instruction Memory
Instruction [31-26]
Instruction [25-21]
Instruction [20-16]
Instruction [15-11]
Instruction [15-0]
Instruction [5-0]
WriteDataSP
Read Register 1 (rs)
Read Register 2 (rt)
Write Register
WriteData
Read Data 1
ReadDataSP
Read Data 2
Register File
FlagZ
FlagS
ALU
ALU Result
Z
Address
Read Data
Data Memory
Write Data
16
Sign Extend
32
ALU Control
SP

ALU



32 bit ALU

For each of the operations (ADD, SUB, AND, OR, XOR, NOT, SLA, SRL, SRA), a sub module for each of the operations is implemented, and in the top module ALU, each of these sub modules are instantiated and assigned the OUT is prepared based on the input ALUfunc opcode.

Each sub module takes A and B as input, assigns temp_out = A op B as output of the sub module.

For example, in the ADD submodule, temp_out is the output and we assign temp_out = A + B.

4. Control Signals

- *[2 bit]* **PCControl**: PCin is enabled when the left bit is set to 1 and disabled otherwise.
  PCout is enabled when the right bit is set to 1 and disabled otherwise.
- **Call**: Set to 1 when a call instruction is received, otherwise is set to 0.
- **ALUop**: Has the value of the opcode and passes it onto the ALU Controller, which uses it as well as the funct value to determine the ALU operation
- *[2 bit]* **RegDst**: Used to choose the destination register from the instruction, with the options being:
  - 0 : rs
  - 1 : rt
  - 2 : rd
- **RegWrite**: Set to 1 to write to a register, otherwise set to 0.
- **ALUSrc1**: Used to choose the source of the first ALU input, with the options being:
  - 0 : ReadData1 (1st read port of Register Bank)
  - 1 : ReadDataSP (read port of SP register)
- **ALUSrc2**: Used to choose the source of the second ALU input, with the options being:
  - 0 : ReadData2 (2nd read port of Register Bank)
  - 1 : Sign Extended Immediate Value
- **SPWrite**: Set to 1 to write to the SP register, otherwise set to 0.
- *[2 bit]* **ZControl**: Zin is enabled when the left bit is set to 1 and disabled otherwise.
  Zout is enabled when the right bit is set to 1 and disabled otherwise.
- *[2bit]* **ZControlSP**: ZSPin is enabled when the left bit is set to 1 and disabled otherwise.
  ZSPout is enabled when the right bit is set to 1 and disabled otherwise.
- **MemtoOut**: Used to choose which data source should be passed onto the write port of the register bank / branch offset for PC
  - 0 : Read Data port of Memory Register
  - 1 : ALU output
- **PCUpdate**: Used to choose the value by which the PC should be incremented, with the options being:
  - 0 : + 4
  - 1 : Sign Extended Immediate Value
- **MemWrite**: Set to 1 to write to Data Memory, otherwise set to 0.
- **WriteDataSrc**: Used to choose the the source to the write port in the Data Memory, with the options being:
  - 0 : ReadData2 (2nd read port of Register Bank)
  - 1 : Updated PC value (Usually used for when some value has to be saved in the stack)
- **SPUpdate**: Used to choose which data source should be passed onto the write port of the SP.
  - 0 : + 4
  - 1 : − 4

5. Control signal generation

opcode	funct	ALU Op	Reg Dst	ALU Src1	ALU Src2	Reg Write	ZControl SP	Mem Write	WriteDataSrc	Memto Out	ZControl	SP Write	SP Update	PC Update	Call
000000	100000	100000	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	100010	100010	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	100100	100100	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	100101	100101	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	100110	100110	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	100111	100111	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	000000	000000	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	000010	000010	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000000	000011	000011	2	0	0	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000001	NA	000001	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000010	NA	000010	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000011	NA	000011	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000100	NA	000100	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000101	NA	000101	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000110	NA	000110	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
000111	NA	000111	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
001000	NA	001000	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
001001	NA	001001	0	0	1	1/0	x	x	x	1/0	10/01/00	x	x	0	0
001010	NA	001010	1	0	1	1/0	x	x	x	0	10/01/00	x	x	0	0
001011	NA	001011	x	0	1	x	x	1/0	0	x	10/01/00	x	x	0	0
001100	NA	001100	1	0	1	1/0	x	x	x	0	10/01/00	x	x	0	0
001101	NA	001101	x	0	1	x	x	1/0	x	x	10/01/00	x	x	0	0
001110	NA	001110	x	x	x	x	x	x	x	x	x	x	x	0	0
001111	NA	001111	x	0	x	x	x	x	x	x	x	x	x	0	0
010000	NA	010000	x	0	x	x	x	x	x	x	x	x	x	0	0
010001	NA	010001	x	0	x	x	x	x	x	x	x	x	x	0	0
010010	NA	010010	x	1	x	x	10/01/00	1/0	0	x	10/01/00	1/0	0	0	0

010011	NA	010011	1	1	x	1/0	10/01/00	x	x	0	10/01/00	1/0	1	0	0
010100	NA	010100	x	1	x	x	10/01/00	1/0	1/0	x	10/01/00	1/0	0	0	0/1
010101	NA	010101	x	1	x	x	10/01/00	x	x	0	10/01/00	1/0	1	1/0	0
010110	NA	010110	1	0	x	1/0	x	x	x	1/0	10/01/00	x	x	0	0
100000	NA	100000	x	x	x	x	x	x	x	x	x	x	x	0	0
100001	NA	100001	x	x	x	x	x	x	x	x	x	x	x	0	0

* The '/' between some values of the control signals indicates that the corresponding control signal takes multiple values during the execution of that instruction, at different timesteps in the given order.
* PCControl Is set to '01' followed by '00' and then '10' for all of the above instructions, at the appropriate timesteps.

6. Micro instructions

i) ADDI, SUBI, ANDI, ORI, XORI, NOTI, SLLI, SRLI, SRAI

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, RegDst =0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ALUSrc2 = 1, ZControl = 10
3	ZControl = 01, MemToOut = 1, RegWrite = 1
4	RegWrite = 0, MemToOut = 0, ZControl = 00
5	PCControl = 10

ii) ADD, SUB, AND, OR, XOR, NOT, SLL, SRL, SRA

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, RegDst =0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ALUSrc2 = 0, ZControl = 10
3	ZControl = 01, MemToOut = 1, RegWrite = 1
4	RegWrite = 0, MemToOut = 0, ZControl = 00
5	PCControl = 10

iii) LD

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, RegDst =1, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ALUSrc2 = 1, ZControl = 10
3	ZControl = 01, MemToOut = 0, RegWrite = 1
4	RegWrite = 0, ZControl = 00
5	PCControl = 10

iv) ST

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ALUSrc2 = 1, ZControl = 10
3	ZControl = 01, WriteDataSrc = 0. MemWrite = 1
4	MemWrite = 0, ZControl = 00
5	PCControl = 10

v) LDSP

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, RegDst =1, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ALUSrc2 = 1, ZControl = 10
3	ZControl = 01, MemToOut = 0, RegWrite = 1
4	RegWrite = 0, ZControl = 00
5	PCControl = 10

vi) STSP

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ALUSrc2 = 1, ZControl = 10
3	ZControl = 01, WriteDataSrc = 0. MemWrite = 1
4	MemWrite = 0, ZControl = 00
5	PCControl = 10

vii) BR

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1)
3	PCControl = 10

viii) BMI, BPL, BZ

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0
3	PCControl = 10

ix) PUSH

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1), SPUpdate = 0, ZControlSP = 10
3	ZControlSP = 01, SPWrite = 1
4	SPWrite = 0, ZControlSP = 00, ALUSrc1 = 1, ZControl = 10, WriteDataSrc = 0
5	ZControl = 01. MemWrite = 1
6	MemWrite = 0, ZControl = 00
7	PCControl = 10

x) POP

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1),ALUSrc1 = 1, ZControl = 10, RegDst = 1
3	ZControl = 01, MemToOut = 0, RegWrite = 1
4	RegWrite = 0, ZControl = 00, SPUpdate = 1, ZControlSP = 10
5	ZControlSP = 01. SPWrite = 1
6	SPWrite = 0, ZControlSP = 00
7	PCControl = 10

xi) CALL

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1), SPUpdate = 0, ZControlSP = 10
3	ZControlSP = 01, SPWrite = 1
4	SPWrite = 0, ZControlSP = 00, ALUSrc1 = 1, ZControl = 10, WriteDataSrc = 1
5	ZControl = 01. MemWrite = 1
6	MemWrite = 0, ZControl = 00, WriteDataSrc = 0, Call = 1
7	PCControl = 10
8	PCControl = 00

xii) RET

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 1, Call = 0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 1, ZControl = 10
3	ZControl = 01, MemToOut = 0, PCControl = 10
4	PCControl = 00, ZControl = 00, PCUpdate = 0, SPUpdate = 1, ZControlSP = 10
5	ZControlSP = 01, SPWrite = 1
6	SPWrite = 0, ZControlSP = 00

xiii) MOVE

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1), ALUSrc1 = 0, ZControl = 10, RegDst = 1
3	ZControl = 01, MemToOut = 1, RegWrite = 1
4	MemToOut = 0, RegWrite = 0, ZControl = 00, PCControl = 10

xiv) HALT

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 00, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1)
3	PCControl = 10

xv) NOP

Time Step	Micro Instructions
1	PCControl = 01
2	PCControl = 10, PCUpdate = 0, Call = 0, ALUop = opcode (from the ISA given in Section 1)

7. Assembler (assembler.py)

A. Instruction Set and Tables:

- The script defines several dictionaries: "opcodes", "funct_codes", and "registers", which map mnemonic instructions, operation codes, function codes, and registers to their binary representations.

B. Input Processing:

- The script takes input from two files: text segment file for instructions and data segment file for data.
- It reads the text segment file to process assembly-like code.
- It strips empty lines, separates labels from instructions, and builds a label table to store the line numbers of labels.

C. Data Processing:

- It reads the data segment file to process declared data and generates a data memory representation.
- It separates data labels and their respective values. The script stores the data in memory and writes it to an output data file.

D. Assembly Process:

- The assembly process translates assembly-like instructions to their binary representation based on the defined ISA.
- For each line of instruction, it translates the assembly mnemonic to the corresponding binary code based on the opcode and function code mappings.
- It interprets the operands of each instruction, the registers, immediate values, and labels.
- It creates the binary representation for each instruction and outputs these machine instructions to a specified output instruction file.

E. Functionality:

- The script handles various types of instructions including arithmetic (ADD, SUB, AND, OR, etc.), immediate operations (ADDI, SUBI, ANDI, etc.), memory operations (LD, ST, LDSP, STSP), branching (BR, BMI, BPL, BZ), stack operations (PUSH, POP), control (CALL, RET), data movement (MOVE), and control instructions (HALT, NOP).

F. Execution:

- The script is designed to be executed from the command line, accepting arguments for the text segment file ("-t"), data segment file ("-d"), and optional output file names for instruction memory ("-oi") and data memory ("-od").