CMPS301 Computer Architecture
Project Report – Phase 1
Team 11

NAME	ID
Rawan Yasser Hashem	1210008
Danah Rafik Hassan	1210221
Basma Mohamed Abd El Hakeem	1210208
Youssef Ahmed Abdelbar	1210090
Nouran Shams Eldeen	1210420

Table of Contents

Op-codes

Instruction	Op-code	Instruction	Op-code	Instruction	Op-code
NOP	**00000**	IADD	**01011**	RET	**10110**
HLT	**00001**	PUSH	**01100**	INT	**10111**
SETC	**00010**	POP	**01101**	RTI	**11000**
NOT	**00011**	LDM	**01110**		
INC	**00100**	LDD	**01111**		
OUT	**00101**	STD	**10000**		
IN	**00110**	JZ	**10001**		
MOV	**00111**	JN	**10010**		
ADD	**01000**	JC	**10011**		
SUB	**01001**	JMP	**10100**		
AND	**01010**	CALL	**10101**		

Instruction Bits Details:

Instruction	16-bit Details					
	15-11	10-8 R1 Read from & maybe write back in	7-5 R2 Read from (two operands)	4-2 Rf Write back in	Index Bit	IMM Bit
NOP	Op-code				Any empty cell is zero	Any empty cell is zero
HLT	Op-code					
SETC	Op-code					
NOT	Op-code	Rsrc1		Rdst		
OUT	Op-code	Rsrc1				
IN	Op-code			Rdst		

MOV	Op-code	Rsrc1		Rdst		
SUB	Op-code	Rsrc1	Rsrc2	Rdst		
AND	Op-code	Rsrc1	Rsrc2	Rdst		
IADD	Op-code	Rsrc1	Rsrc2	Rdst		IMM bit
	IMM	IMM	IMM	IMM	IMM	IMM
PUSH	Op-code	Rsrc1				
POP	Op-code			Rdst		
LDM	Op-code			Rdst		IMM bit
	IMM	IMM	IMM	IMM	IMM	IMM
LDD	Op-code	Rsrc1		Rdst		IMM bit
	IMM	IMM	IMM	IMM	IMM	IMM
STD	Op-code	Rsrc1	Rsrc2			IMM bit
	IMM	IMM	IMM	IMM	IMM	IMM
JZ	Op-code	Rsrc1				

JN	Op-code	Rsrc1				
JC	Op-code	Rsrc1				
JMP	Op-code	Rsrc1				
CALL	Op-code	Rsrc1				
RET	Op-code					
INT	Op-code				Index	
RTI	Op-code					

Control Signals:

INST	Rd Rs Sel	REGW	PCDIS	IMM bit	MW	MR	MTRslctr	SP+	SP-	ALUSLCTR	OUTEN	JMP	Z	N	C	INT	RET	FR	FLAGEN	UC	Rsrc1/ Rsrc2 Sel	Write Data Select	address
NOP	0	0	0	0	0	0	00	0	0	00	0	0	0	0	0	0	0	0	0	0	0	0	0
HLT	0	0	1	0	0	0	00	0	0	00	0	0	0	0	0	0	0	0	0	0	0	0	0
SETC	0	0	0	0	0	0	00	0	0	011	0	0	0	0	1	0	0	0	1	0	0	0	0
NOT	1	1	0	0	0	0	00	0	0	000	0	0	0	0	0	0	0	0	1	0	0	0	0

INC	**1**	**1**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**1 1 1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
OUT	**0**	**0**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**1 0 0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
IN	**1**	**1**	**0**	**0**	**0**	**0**	**1 0**	**0**	**0**	**0 0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
MOV	**1**	**1**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**1 0 0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
ADD	**1**	**1**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**1 1 0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
SUB	**1**	**1**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**0 1 0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**

AND	1	1	0	0	0	0	00	0	0	001	0	0	0	0	0	0	0	0	1	0	0	0	0
INST	**RdRs Sel**	**REGWRITE**	**PCDISABLE**	**IMMbit**	**MW**	**MR**	**MTRslctr**	**SP+**	**SP-**	**ALUslctr**	**W_EN OUT**	**JMP**	**Z**	**N**	**C**	**INT**	**RET**	**FR**	**FLAGEN**	**UC**	**Rsrc 1/ Rsrc 2 Sel**	**Write Data Select**	**Exc Cau F**
IADD	1	1	0	1	0	0	00	0	0	110	0	0	0	0	0	0	0	0	1	0	0	0	0
PUSH	0	0	0	0	1	0	00	0	1	00	0	0	0	0	0	0	0	0	0	0	0	0	1
POP	1	1	0	0	0	1	01	1	0	00	0	0	0	0	0	0	0	0	0	0	0	0	1

LDM	**1**	**1**	**0**	**1**	**0**	**0**	**0 0**	**0**	**0**	**1 0 1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
LDD	**1**	**1**	**0**	**1**	**0**	**1**	**0 1**	**0**	**0**	**1 1 0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**
STD	**0**	**0**	**0**	**1**	**1**	**0**	**0 0**	**0**	**0**	**1 1 0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**
JZ	**0**	**0**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**0 0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
JN	**0**	**0**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**0 0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
JC	**0**	**0**	**0**	**0**	**0**	**0**	**0 0**	**0**	**0**	**0 0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

JMP	0	0	0	0	0	0	00	0	0	00	0	1	0	0	0	0	0	0	0	1	0	0	0
CALL	0	0	0	0	1	0	00	0	1	100	0	1	0	0	0	0	1	0	0	0	0	1	1
RET	0	0	0	0	0	1	01	1	0	00	0	1	0	0	0	0	1	0	0	0	0	0	1
INST	**Rd Rs Sel**	**REGWRITE**	**PCDISABLE**	**IMM bit**	**MW**	**MR**	**MTRslctr**	**SP+**	**SP-**	**ALUslctr**	**OUTEN**	**JMP**	**Z**	**N**	**C**	**INT**	**RET**	**Restored**	**FLAGEN**	**UC**	**Rsrc 1/ Rsrc 2 Sel**	**Write Data Select**	**Exc Caufet**
INT	0	0	0	0	1	0	00	0	1	00	0	1	0	0	0	1	1	0	0	0	0	1	1
RTI	0	0	0	0	0	1	01	1	0	00	0	1	0	0	0	0	1	1	0	0	0	0	1

ALU selector	Operation	ALU selector	Operation
000	**NOT**	100	**NOP (Data1)**
001	**AND**	101	**NOP (Data2)**
010	**SUB**	110	**ADD**
011	**SETC**	111	**INC**

Control Signals Explained:

Signal	Explanation
RdRs Sel	**Controls which register to write back in (Rdst or Rsrc1)** **0 -> Rsrc1** **1 -> Rdst**
REG WRITE	**Controls write enable of register file in chosen destination.**
PC DISABLE	**Freezes the PC**
ALU SRC	**Controls input2 for the ALU.** **0 -> Rsrc2** **1 -> IMM**
MW	**Controls write enable of data memory.**
MR	**Controls read enable of data memory.**
MTR SLCTR	**Controls the source of data written back (ALU result or memory data).** **00 -> ALU result**

	01 -> Memory out **10 -> IN-Port value**
ALU SLCTR	**Controls the operation performed by ALU.**
SP+	**Increments SP when POP.**
SP-	**Decrements SP when PUSH.**
SP- && SP+ (memory address input MUX selector)	**Controls which address to read/write (effective address passed from ALU or SP location).** **10 -> EFF address (ALU result)** **00 -> SP without change (POP)** **01 -> new SP (decremented) (PUSH)**
OUT EN	**Controls write enable of output port.**
JMP	**Indicates a branch instruction, controls the branch handler unit.**

	JMP = 0 in RET & RTI as they cannot be handled before memory stage, so we handle them as normal instruction then flush the previous registers (3 stalls).
Z	**Indicates JZ (conditional branch), controls the branch handler unit**
N	**Indicates JN (conditional branch), controls the branch handler unit**
C	**Indicates JC (conditional branch), controls the branch handler unit**
INT	**Controls the reservation of the flags, interrupt the IM to get the instruction out at location Index + 3 and put it in the PC.**
RET	**Indicates an instruction that changes the PC and stall 3 times.**
Flag enable	**Controls write enable for flag register CCR.**
FR	**Flag Restored Bit controls the restoring to flag register CCR.**
UC	**Indicates an unconditional branch, does not access branch prediction unit.**
Rsrc1/Rsrc2 Sel	**Control the first input to the ALU to choose Rsrc2 in STD**

Pipeline Stages Design:

1) Fetch:

The processor fetches the next instruction from the instruction cache address pointed to by the program counter (PC).

After fetching the instruction, the PC is incremented to point to the next instruction in memory, if it wasn't a branch instruction.

2) Decode:

The processor decodes the fetched instruction to determine the operation it represents and the operands involved.

If the instruction involves register operands, the decode stage reads the values from the specified registers.

In addition, control signals are generated to control the execution of the instruction in a correct way.

3) Execute:

The processor performs the actual logical or arithmetic operations in the ALU, using the operands obtained from the decode stage, according to the instruction.

If the instruction involves memory operations, it calculates the required memory address.

Updates the flag register and determines whether the zero branch should be taken or not.

4) Memory:

The processor performs the memory access operation. This includes reading data from memory for load instructions or writing data to memory for store instructions.

5) Write Back:

This stage receives the result of the instruction execution from the execute or memory stage, depending on the type of instruction.

If the instruction involves updating registers, the write back stage updates the specified destination register(s) with the computed result.

Once the result has been written back to the destination, the instruction is considered complete.

Pipeline Registers Details:

Register	Size	Stored Values	Stored Control Signals	Connected Components (IN)	Connected Components (OUT)
IF/ID	64 bit	• Current instruction • IMM • Next instruction • PC • IN port value		• Instruction memory • IN port • Flush Unit • PC	• Control Unit • Register File • ID/EX register • Hazard detection unit
ID/EX	103 bit	• PC+1 • IN port value • PC • Index • Rdst • Rsrc1 • Data 1 • Data 2	• Flag enable • OUT enable • REG WRITE • INT • MR • MW • MTR • SP+/SP- • FR • Write Data Selector • RET • Z • N • C • ALU Selector	• Register file • IF/ID register • Control Unit • Flush Unit	• ALU • ALU SRC mux • Rsrc1-Rsrc2 mux • Rd-Rs mux • EX/MEM register • Hazard Detection Unit • Exception detection unit

			• ALU SRC • JMP • UC • Rd-Rs sel • Rsrc1/Rsrc2 sel		
EX/MEM	103 bit	• Data 1 • Data 2 • Flags • ALU result • IN port value • DST Reg • PC • PC+1 • Index	• OUT enable • REG WRITE • INT • MR • MW • MTR • SP+/SP- • Write Data Selector • RET	• ALU • Flag Register CCR • Rd-Rs mux • ID/EX register • Flush Unit	• Data memory • Write data mux • Address mux • MEM/WB register • Hazard Detection Unit
MEM/WB	68 bit	• SP Value • ALU result • MemOut • Index • DST Reg • IN port value	• OUT enable • REG WRITE • INT • MTR • RET	• EX/MEM register • Flush Unit • SP register • Data memory	• MTR mux • PC INT signal mux • Register file • SP update unit

Hazards:

Type of Hazard	Source of Hazard	Handling	
		Detection	**Solution**
Data Hazard	**Load-use**	**Hazard Detection Unit:** Checks if 1. MemRead = 1 in ID/EX 2. Rdst in ID/EX = Rsrc1 in IF/ID 3. Rdst in ID/EX = Rsrc2 in IF/ID (1&&2) or (1&&3) -> Load-Use Hazard detected.	**Forwarding Unit:** Stall twice (NOP) until data is read from memory. Forward ReadData from MEM/WB to EXECUTE stage. **LDD & POP instructions.**
	Read after write	**Hazard Detection Unit:** Checks if 1. RegWrite_1 = 1 & MemRead = 0 in EX/MEM 2. Rdst in EX/MEM = Rsrc1 in ID/EX 3. Rdst in EX/MEM = Rsrc2 in ID/EX (1&&2) or (1&&3) -> Read after write Hazard detected.	**Forwarding Unit:** If (IN || LDM), Forward data accordingly: • IN -> IN-Port value • LDM -> Immediate value in ID/EX Else, forward WriteData from ALU result to EXECUTE stage.

Control Hazard	**Branching**	**Static prediction (untaken)** Any jump will be untaken till the execution stage. If we know in execution that the branch should be taken, we will flush two instructions and jump (PC = Rsrc1)

Causes of Exceptions:

Types	Causes
Overflow	INC – ADD – ADDI- SUB – PUSH – POP – CALL – RET – RTI – PC might overflow every new cycle
Out of bounds	LDD – STD – JZ – JC – JMP – CALL – RET – RTI

Elaboration:

INC:
R[Rsrc1] + 1 overflows → Execution stage
ADD:
Overflow → Execution stage
ADDI:
Overflow → Execution stage
SUB:
Overflow → Execution stage
PUSH:
SP -= 1 overflows → Memory stage
DataMemory[SP] out of bound → Memory stage (after seeing SP value, if it is greater than 2^12 – 1, then out of bound)
POP:
SP += 1 overflows → Memory stage
DataMemory[SP] out of bound → Memory stage
LDD:
DataMemory[R[Rsrc1]] out of bound, as Memory has 2^12 addresses, but

R[Rsrc1] can go up to 2^16-1 → Decode stage (after seeing R[Rsrc1] value, if it is greater than 2^12 – 1, then out of bound)

STD:

DataMemory[R[Rsrc]] out of bound → Decode stage

JZ:

PC = R[Rdst] may be out of bound, if instruction cache size is smaller than 2^16-1 → Execution stage

JN:

PC = R[Rdst] out of bound → Execution stage

JC:

PC = R[Rdst] out of bound → Execution stage

JMP:

PC = R[Rdst] out of bound → Decode stage (when decoding, we know it is a jump command, and we also have the value of R[Rdst], if it is greater than the size of PC, then out of bound) (unlike JZ and JC, we do not need to wait till the execution stage to check whether we will actually jump or not, as in the JPM command, we will unconditionally jump)

CALL:

DataMemory[SP] out of bound, as memory has 2^12 addresses, but PC can go up to 2^16-1 → Memory stage

PC + 1 overflows → Fetch stage

SP -=1 overflows → Memory stage

RET:

SP += 1 overflows → Memory stage

DataMemory[SP] out of bound → Memory stage

RTI:

SP += 1 overflows → Memory stage

DataMemory[SP] out of bound → Memory stage

After each fetch, PC + 1 might overflow → Fetch stage

SCHEMATIC DIAGRAMS LINK: https://drive.google.com/drive/folders/1UCywb94f2ysVs7gsQyVfLdK4qYGf1CYo