Cairo University
Faculty of Engineering

Credit Hours System

Computer Architecture
CMPN301

Computer Architecture Project
Phase 1 Report

Team Members:

Ahmed Magdy Abdelaziz	**1190200**
Mahmoud Khaled Mahmoud	**1190141**
Ziad Ezzat Sedky	**1190032**
Ahmed Mohamed Saad Hussein	**1190184**

Instructions Sequence:

Instructions divided into 4 categories:

- Instructions with no operand
- Instructions with only one operands
- Instructions with only two operands
- Instructions with three operands

1) **Instructions with no operand:**
   Divided into 3 groups as following:
   - CLRC , SETC → For carry bit
   - RET , RTI → For Memory (SP to PC)
   - NOP
2) **Instructions with one operand:**
   Divided into 2 Groups as following:
   1) **Read from register file and this divided into 4 Groups:**
      - OUT → Destination is OUT port
      - PUSH → Destination is Memory
      - JMP , JZ , JC → Destination is PC
      - CALL → Destination is PC and Memory
   2) **Write into register file and this divided into 2 Groups:**
      1) IN → Source is IN Port
      2) POP → Source is Memory
3) **Instructions with two operands:**

   Divided into 4 groups as following:

   1) **Memory:**
      - LOAD
        1. LDM → SRC will be immediate value
        2. LDD → SRC will be any Register

- Store → STD

2) **MOV**
3) **NOT**
4) **INC or DEC**

4) Instructions with Three operands:

Divided into 4 Groups as following:

1) **Adding :**
   - ADD → SRC2 will be from Register file
   - IADD → will read from Immediate value
2) **SUB**
3) **AND**
4) **OR**

Instruction details:

Total Instruction → 32 bits

RDST & RSRC1 & RSRC2 → 3-bits , X → Don't Care , IMM (16 bits immediate value)

NOTE) Colored bits means → will be read from Register File

1) **NOP → 00 RDST RSRC1 RSRC2 X 00 XX IMM**
2) **SETC → 00 RDST RSRC1 RSRC2 0 01 XX IMM**
3) **CLRC → 00 RDST RSRC1 RSRC2 1 01 XX IMM**
4) **RET → 00 RDST RSRC1 RSRC2 0 10 XX IMM**
5) **RTI → 00 RDST RSRC1 RSRC2 1 10 XX IMM**

   [31:30] → Identify one of four categories that mentioned above
   [19:18] → Select one operation from the selected category

[20] → Used only for Identify which operation will be done on the carry (SET , CLEAR) and for RTI and RET to identify restoring of PC only or PC and Flag registers

---

31:30 rd:rs1:rs2 20: 19:18 17:16

6) OUT → 01 RDST RSRC1 RSRC2 0 00 XX IMM

7) PUSH → 01 RDST RSRC1 RSRC2 0 01 XX IMM

8) JZ → 01 RDST RSRC1 RSRC2 0 10 10 IMM

9) JC → 01 RDST RSRC1 RSRC2 0 10 01 IMM

10) JMP → 01 RDST RSRC1 RSRC2 0 10 11 IMM

11) CALL → 01 RDST RSRC1 RSRC2 0 11 11 IMM

12) IN → 01 RDST RSRC1 RSRC2 1 00 XX IMM

13) POP → 01 RDST RSRC1 RSRC2 1 01 XX IMM

[31:30] → Select one of Categories mentioned above

[20] → Indicate Write operation or Read operation

[19:18] → Select one Operation from this Category

[17:16] → Not be used for all except JMP to Select which JMP

NOTE: IN/OUT has same [19:18] Select between them by bit 20

---

14) MOV → 10 RDST RSRC1 RSRC2 X 00 0X IMM

15) NOT → 10 RDST RSRC1 RSRC2 X 01 0X IMM (ALU)

16) INC → 10 RDST RSRC1 RSRC2 0 10 0X IMM (ALU)

17) DEC → 10 RDST RSRC1 RSRC2 1 10 0X IMM (ALU)

18) LDM → 10 RDST RSRC1 RSRC2 0 11 1X IMM (ALU + Imm)

19) LDD → 10 RDST RSRC1 RSRC2 0 11 0X IMM

20) STD → 10 RDST RSRC1 RSRC2 1 11 0X IMM

[31:30]→ Select one of four categories mentioned above

[19:18]→ Select which operation from this category

Notes: INC/DEC have same [19:18] select Add or sub by bit 20 and

LDM/ LDD have same [19:18] we will select IMM or Reg by anding bit [17] with [31] if 1 → REG and else IMM value (16-bits) , Store also has same [19:18] as LDM/LDD but we distinct it from LOAD operations by bit [20]

---

21) ADD → 11 RDST RSRC1 RSRC2 X 00 0X IMM (ALU)

22) IADD → 11 RDST RSRC1 RSRC2 X 00 1X IMM (ALU + Imm)

23) SUB → 11 RDST RSRC1 RSRC2 X 01 0X IMM (ALU)

24) AND → 11 RDST RSRC1 RSRC2 X 10 0X IMM (ALU)

25) OR → 11 RDST RSRC1 RSRC2 X 11 0X IMM (ALU)

[31:30]→ Select one of four categories mentioned above

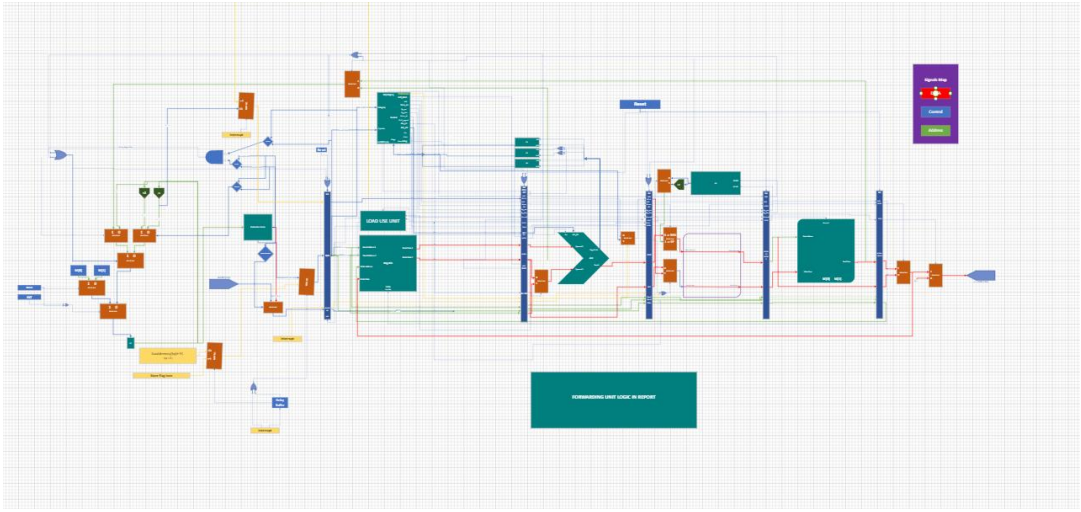
[19:18]→ Select one operations from this category

Notes: ADD/IADD have same [19:18] and we will difference between them by bit [17] anding with bit 31 and if 1 → IMM value will be used , else → REG will be used

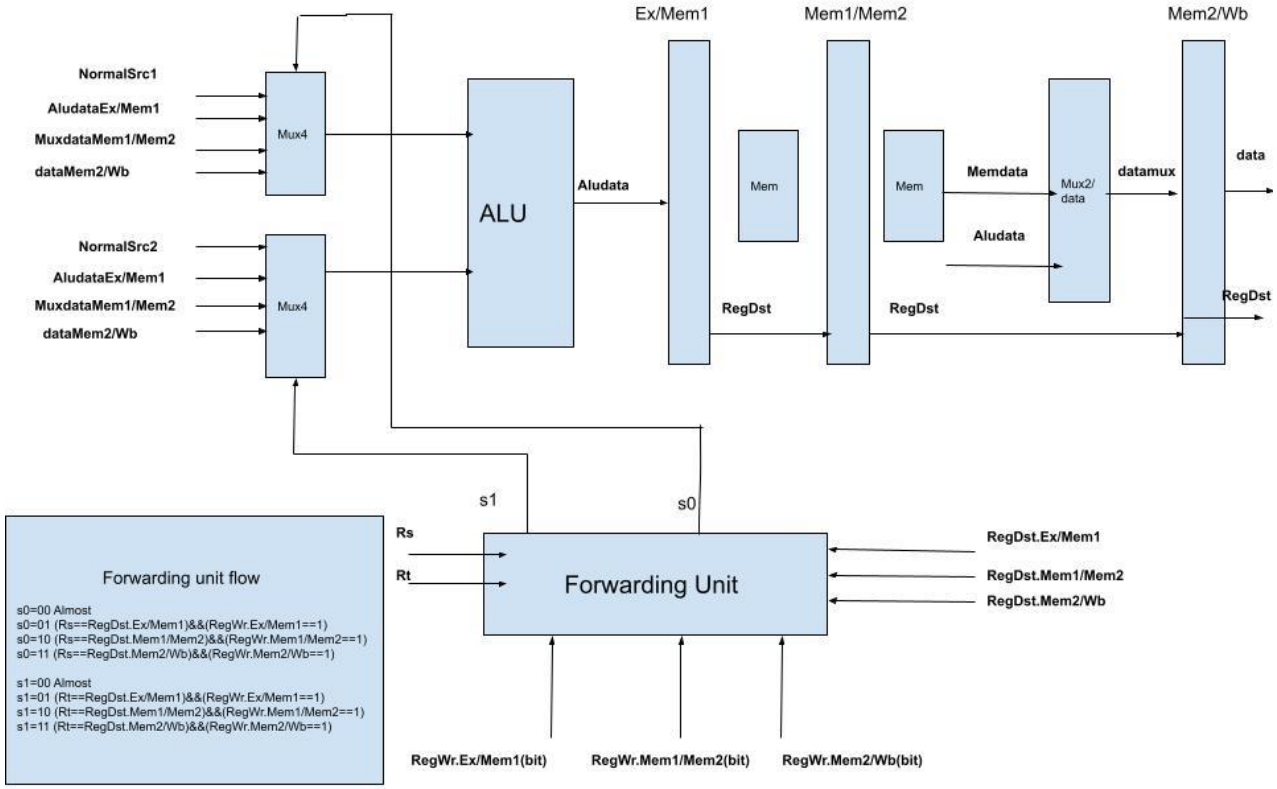
INST	JMP_FLAG	WB	Mem_src	Sp_inc	Sp_dec	Mem_W	Alu_src
NOP	0	0	0	0	0	0	0
SETC	0	0	0	0	0	0	0
CLRC	0	0	0	0	0	0	0
CALL	1	0	1	1	0	1	0
JZ	depends->ZF	0	0	0	0	0	0
JC	depends->CF	0	0	0	0	0	0
JMP	1	0	0	0	0	0	0
RET	0	0	1	1	0	0	0
RTI	0	0	1	1	0	0	0
POP	0	1	1	1	0	0	0
PUSH	0	0	1	0	1	0	0
IN	0	1	0	0	0	0	1
OUT	0	0	0	0	0	0	0
STD	0	0	0	0	0	1	0
IADD	0	1	0	0	0	0	1
LDM	0	1	0	0	0	0	0
LDD	0	1	0	0	0	0	0
ADD	0	1	0	0	0	0	0
MOV	0	1	0	0	0	0	0
NOT	0	1	0	0	0	0	0
INC	0	1	0	0	0	0	0
DEC	0	1	0	0	0	0	0
SUB	0	1	0	0	0	0	0
AND	0	1	0	0	0	0	0
OR	0	1	0	0	0	0	0

INST	Alu_OP			Cin	M2R	SETC/CLRC		RET/RTI		Call	OUT_signal
NOP	1	0	0	0	0	0	0	0	0	0	0
SETC	1	0	0	0	0	1	1	0	0	0	0
CLRC	1	0	0	0	0	1	0	0	0	0	0
CALL	1	0	0	0	0	0	0	0	0	1	0
JZ	1	0	0	0	0	0	0	0	0	0	0
JC	1	0	0	0	0	0	0	0	0	0	0
JMP	1	0	0	0	0	0	0	0	0	0	0
RET	1	0	0	0	0	0	0	1	0	0	0
RTI	1	0	0	0	0	0	0	1	1	0	0
POP	1	0	0	0	1	0	0	0	0	0	0
PUSH	1	0	0	0	0	0	0	0	0	0	0
IN	1	0	0	1	0	0	0	0	0	0	0
OUT	1	0	0	0	0	0	0	0	0	0	1
STD	1	0	0	0	0	0	0	0	0	0	0
IADD	0	0	0	0	0	0	0	0	0	0	0
LDM	1	0	0	1	0	0	0	0	0	0	0
LDD	1	0	0	0	1	0	0	0	0	0	0
ADD	0	0	0	0	0	0	0	0	0	0	0
MOV	1	0	0	0	0	0	0	0	0	0	0
NOT	1	0	1	0	0	0	0	0	0	0	0
INC	0	0	1	0	0	0	0	0	0	0	0
DEC	0	1	0	0	0	0	0	0	0	0	0
SUB	0	1	1	0	0	0	0	0	0	0	0
AND	1	1	0	0	0	0	0	0	0	0	0
OR	1	1	1	0	0	0	0	0	0	0	0

Schematic



Design Link: [DESIGN]



LOAD USE UNIT LOGIC:

```
LOAD <= '1'
WHEN ((Dec_Exec_MemRead='1'and((Dec_Exec_Rt=Fet_Dec_Rs) or
(Dec_Exec_Rt=Fet_Dec_Rt))))
 ELSE '1'    WHEN
((Exec_Mem1_MemRead='1'and((Exec_Mem1_Rt=Fet_Dec_Rs) or
(Exec_Mem1_Rt=Fet_Dec_Rt))))
--  ELSE '1'    WHEN (Result='1')
 Else '0';
```

CHANGES DONE TO DESIGN:

- Branching Logic moved from decode to execute stage to work with forwarding unit (placed also in execute Stage)
- Structural Hazard Detection unit moved from fetch to decode stage.
- Interrupt logic Changed.

Pipeline Registers Details

IF/ID REG (48 BITS):

INPUT	BITS	**47-32**	**31-16**	**15-0**
	Signal	PC+1	INSTRUCTION	IMM/IN

ID/IE REG (88 BITS):

Input	BITS	87-73	72-57	56-54	53-51	50-48	47-32	31-16	15-0
	SIGNAL	Controller Signals	PC + 1	RSRC1 ADD	RSRC2 ADD	RDST ADD	RSRC1 Value	RSRC2 Value	IMM OR IN

Controller Signals:

SET_CLEAR <= Decode_Buffer_OUT(87 downto 86);
Write_back <= Decode_Buffer_OUT(85);
MEM_SRC <= Decode_Buffer_OUT(84);
SP_INC <= Decode_Buffer_OUT(83);
SP_DEC <= Decode_Buffer_OUT(82);
MEM_WRITE <= Decode_Buffer_OUT(81);
Out_Signal <= Decode_Buffer_OUT(80);
ALU_SRC <= Decode_Buffer_OUT(79);
ALU_Operation <= Decode_Buffer_OUT(78 downto 76);
CIN_Signal <= Decode_Buffer_OUT(75);
CALL_Signal <= Decode_Buffer_OUT(74);
MEM_TO_REG <= Decode_Buffer_OUT(73);

IE/MEM1 REG (65 BITS):

Input	BITS	64-49	48-41	40-25	24-9	8-6	5-3	2-0
	SIGNAL	PC+1	Controller Signals	ALU_RESULT	RSRC2_VALUE	Rsrc1_add	Rsrc2_add	Rdst_add

Controller Signals:

Write_back <= EX_MEM1_Buffer(48);
MEM_SRC <= EX_MEM1_Buffer (47);
SP_INC <= EX_MEM1_Buffer (46);
SP_DEC <= EX_MEM1_Buffer (45);
MEM_WRITE <= EX_MEM1_Buffer (44);
Out_Signal <= EX_MEM1_Buffer (43);
CALL_Signal <= EX_MEM1_Buffer (42);
MEM_TO_REG <= EX_MEM1_Buffer (41);

Mem1/MEM2 REG (64 BITS):

Input	BITS	63-48	47-41	40-25	24-9	8-6	5-3	2-0
	SIGNAL	PC+1	Controller Signals	READ_ADD	WRITE_DATA	Rsrc1_add	Rsrc2_add	Rdst_add

Controller Signals:

WB_OUT <= BUFF_OUT(47);
MEM_TO_REG_OUT <= BUFF_OUT(46);
SP_INC_OUT <= BUFF_OUT(45);
SP_DEC_OUT <= BUFF_OUT(44);
MEMW_OUT <= BUFF_OUT(43);
OUT_SIG_OUT <= BUFF_OUT(42);
CALL_SIG_OUT <= BUFF_OUT(41);

MemSt2/Wb REG (44 BITS):

Input	BITS	43	42	41	40-25	24-9	8-6	5-3	2—0
	SIGNAL	Wb	M2r	outEn	Memst2	Memst1	RSRC1 Value	RSRC2 Value	Rdst