Cairo University
Faculty of Engineering
CMPN301
Spring 2022

Computer Architecture project

Phase 01 documentation

Team - 4

Khadija Swelam **1180377**

Habiba Assem **1180450**

Nada Tarek **1180504**

Muhab Hossam **1180074**

Table of Contents

Instructions details

	OPCODE	RDST	RSRC1	RSRC2	IMM	
	5 bits	3 bits	3 bits	3 bits	16 bits	
NOP	00000	x	x	x	x	
HLT	00001	x	x	x	x	
SETC	00010	x	x	x	x	
RET	00011	x	x	x	x	
RTI	00100	x	x	x	x	
SINGLE OPERAND	5 bits	3 bits	3 bits	3 bits	16 bits	
PUSH	01000	Rdst	x	x	x	
POP	01001	Rdst	x	x	x	
OUT	01010	Rdst	x	x	x	
IN	01011	Rdst	x	x	x	
CALL	01100	x	x	x	IMM	
INT	01101	1/0	x	x		1 bit for the index
INC	01110	Rdst	x	x	x	
NOT	01111	Rdst	x	x	x	
TWO OPERAND	5 bits	3 bits	3 bits	3 bits	16 bits	
MOV	10000	Rdst	Rsrc1	x	x	
SWAP	10001	Rdst	Rsrc1	x	x	
ADD	10010	Rdst	Rsrc1	Rsrc2	x	
SUB	10011	Rdst	Rsrc1	Rsrc2	x	
AND	10100	Rdst	Rsrc1	Rsrc2	x	
JUMPS	5 bits	3 bits	3 bits	3 bits	16 bit	
JZ	11000	x	x	x	IMM	
JN	11001	x	x	x	IMM	
JC	11010	x	x	x	IMM	
JMP	11011	x	x	x	IMM	
MEMORY	5 bits	3 bits	3 bits	3 bits	16 bit	
IADD	11100	Rdst	Rsrc1	x	IMM	
LDM	11101	Rdst	x	x	IMM	
LDD	11110	Rdst	Rsrc1	x	OFFSET	
STD	11111	Rdst	Rsrc1	x	OFFSET	

Control Signals

The control signals can be found in the excel file included in the submission

Control unit

Description of control signals

ccr_wr_en	Bit 0 – Carry flag write enable Bit 1 – Negative flag write enable Bit 2 – Zero flag write enable

stack_en	Enable signal for instructions that changes the stack pointer

alu_imm	Immediate value source IN port – 0 IMM EA – 1

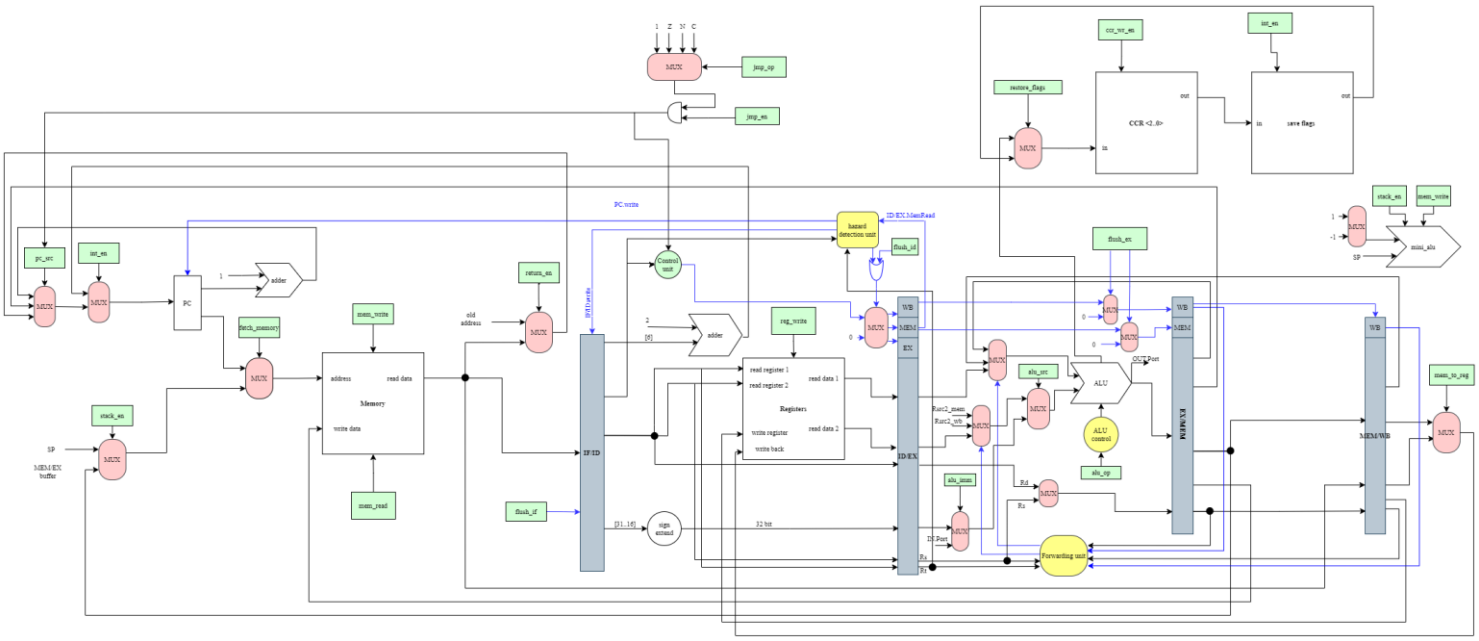
jmp_op	JMP operations type JZ – 00 JN – 01 JC – 10 JMP – 11

Pipeline Registers Contents and sizes

Register	IF/ID	ID/EX	EX/MEM	MEM/WB
Stored control signals [1 bit each]		EX + MEM + WB	MEM + WB	WB
Stored values	Instruction bits [30] flush_if [1]	reg1 data [32] reg2 data [32] Rsrc1 (Rs) [3] Rsrc2 (Rt) [3] Rd [3] IMM EA [32]	ALU [32] Rd/Rs [3]	MEM [32] ALU [32] Rd/Rs [3]

Schematic

Check clear version (.png file) in submission file



Pipeline hazards

Structural hazards

Structural hazards arise from the need to use the same component at the same time, due to the Von Neumann architecture there is a structural hazard between the Fetch and Memory stages as there is only one memory for instructions and data.

This was considered by the hazard detection unit which detects if there is a control signal for the memory stage, in this case we stall the pipeline by continuously fetching the already fetched address without using the memory, this is done by the pc_write signal, either enable PC write or disable it.

Data hazards

This hazard arises from the dependencies of some instruction on the result of a previous instruction, we solved this by implementing full forwarding.

This allows forwarding of the data from the ALU or the memory stages directly without the need to stall the pipelining until data is available after the WriteBack stage. To detect a hazard in the Ex stage, we compare the EX/MEM destination register against both ID/EX source registers that has been read which are all passed to the forwarding unit, If either comparison condition is true, we forward the result from the prior ALU to the next ALU as an operand. If the condition is false, then the operands are passed from the register files. Similarly, for the second path, we check MEM/WB destination register against both ID/EX source registers. If the condition is true, the result value stored in the Mem/Wb register will be forwarded.

Load-Use hazards

For Load-use hazards, which arise when an instruction that comes after a load instruction depends on the data to be loaded, in this case we cannot forward the data from the ALU or the memory as it is only available after the WriteBack stage, the hazard detection unit identifies that this instruction requires a stall and inserts a bubble by zeroing the ID/EX register and its control signals.

The hazard detection unit detects that it is a load instruction not only by its OPCODE, but by using the mem_to_reg control signal which is only equal to 1 in case of load instructions, and if the destination register for that instruction is equal to one of the operands of the next instruction, then there is a load-use hazard.

Control Hazards

Control hazards arises when there is a branching or jumps instructions, so the currently fetched instruction is not the one to be executed. Depending on the jump condition, the next instruction may or may not be executed so to solve this we assumed an Always-not-taken static prediction therefore we always execute the next instruction until the condition flags are checked and it is confirmed that the jump is taken, then the control unit sends signals to flush the IF/ID, ID/EX and EX/MEM registers.