**Team #18**

# **Design Report**

|  |  |  |
| --- | --- | --- |
| **Name** | **BN** | **SEC** |
| **Abdelaziz Salah Mohammed** | **1** | **2** |
| **Moaz Mohammed** | **24** | **2** |
| **Mohammed Nabil** | **19** | **2** |
| **Mahmoud Reda** | **21** | **2** |

**Delivered To:**

**DR: Mayada Hadhoud**

12/4/2022

**Instructions format:**

**1- 16 bits instructions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Instruction | Opcode  (5 bits) | Rdst  (3 bit) | Rsrc  (3 bit) | Remaining  (5 bit) |
| NOT Rdst | 00000 | Rdst | 000 | 00000 |
| INC Rdst | 00001 | Rdst | 000 | 00000 |
| DEC Rdst | 00010 | Rdst | 000 | 00000 |
| MOV Rdst, Rsrc | 00011 | Rdst | Rsrc | 00000 |
| ADD Rdst, Rsrc | 00100 | Rdst | Rsrc | 00000 |
| SUB Rdst, Rsrc | 00101 | Rdst | Rsrc | 00000 |
| AND Rdst, Rsrc | 00110 | Rdst | Rsrc | 00000 |
| OR Rdst, Rsrc | 00111 | Rdst | Rsrc | 00000 |
| SHL Rdst, Imm | 01000 | Rdst | 000 | Imm |
| SHR Rdst, Imm | 01001 | Rdst | 000 | Imm |
| PUSH Rsrc | 01010 | 000 | Rsrc | 00000 |
| POP Rdst | 01011 | Rdst | 000 | 00000 |
| LDD Rdst, Rsrc | 01101 | Rdst | Rsrs | 00000 |
| STD Rdst, Rsrc | 01110 | Rdst | Rsrc | 00000 |
| JZ Rdst | 01111 | Rdst | 000 | 00000 |
| JN Rdst | 10000 | Rdst | 000 | 00000 |
| JC Rdst | 10001 | Rdst | 000 | 00000 |
| JMP Rdst | 10010 | Rdst | 000 | 00000 |
| CALL Rsrc | 10011 | 000 | Rsrc | 00000 |
| RET | 10100 | 000 | 000 | 00000 |
| RTI | 10101 | 000 | 000 | 00000 |
| OUT Rsrc | 10111 | 000 | Rsrc | 00000 |
| IN Rdst | 11000 | Rdst | 000 | 00000 |
| NOP | 11001 | 000 | 000 | 00000 |
| SETC | 11010 | 000 | 000 | 00000 |
| CLCR | 11011 | 000 | 000 | 00000 |

**2- 32 bits instructions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction | Opcode  (5 bits) | Rdst  (3 bits) | Rsrc  (3 bits) | Remaining  (5 bits) | Imm  (16 bits) |
| LDM Rdst, Imm | 11100 | Rdst | 000 | 00000 | Imm |

## **Pipeline Registers:**

|  |  |  |  |
| --- | --- | --- | --- |
| Register Name | Size  (bits) | Input | Connected to |
| PC | 32 | MUX1 Result | Instruction Memory |
| F2D Buffer | 17(16 instruction + 1 IF/ID Flush) | Instruction Memory Result | Decode Stage |
| Reg File | R[0:7]<15:0> | Instruction[10:8]  Dest, Instruction[7:5]  Src,  Write Back Data, Write Back Address, WB Signal | D2E Buffer Register |
| detectLoad | 1 | Hazard Detection Unit | Hazard Detection Unit |
| st, sst | 2 | Control Unit | ALU Second operand MUX |
| D2E Buffer | 52 ( 3 bits register dest + 3 bits src address + 16 bits rd1 + 16 bits rd2 + 14 bits CU signals) | Decode Stage | Execute Stage |
| CCR(Condition Code Register) | 4 | ALU | E2M Buffer |
| E2M | 45 (6 bits CU signals + 4 CCR + 16 bits ALU out + 16 bits read data 2 from registers + 3 bits register dest) | Execute Stage | Memory Stage |
| M2W | 39 ( 4 bits signals + 16 bits ALU out + 16 bits Data Memory Out + 3 bits register dest) | Memory Stage | Write back Stage |

## **32 bits instruction manipulation:**

1- If opcode = 32 bit instr opcode(LDM):

first cycle:

CU:

1- put st, sst = 11

2- make the signals to be nop signals(bubble)

ALU: will do nop instr

second cycle:

CU:

if st, sst == 11:

put sst = 0, st = 1

make signals of LDM

ALU: will take the immediate as the second operand because st = 1

3rd cycle:

CU: if st == 1 && current opcode != 32 bit instr opcode:

st = 0

else:

the same as first cycle (new instr)

## **Hazard Detection:**

Takes opcode of current instruction && CCR

**1- Data hazards:**

* **Solution: Full Forwarding unit:**

**Responsible for determining source data for ALU operands by generating 2 selectors:**

Forwarding selector1 & 2,

If register src address of the operand == execute stage register dest && we will do write back, then we will forward data from EX to EX by making the selector = 2

Else If register src address of the operand == M2W stage register dest && we will do write back, then we will forward data from MEM to EX by making the selector = 1

* **Solution: Load Use Hazard:**

First cycle: If opcode == LDM opcode: make detectLoad = 1 & prevent incrementing the PC

Second cycle: if detectLoad = 1: raise bubble signal to CU, make the pcSrc to take the decrement of the PC(to make it Fetch the instruction after the LDM again)

**2- Control hazards(Branch):**

If the CCR bits are formed to branch, so raise the selector of the MUX between CU and D2E buffer to make the result signals 0 to flush the Decode stage, and raise IF\ID Flush bit in order to make the Decode stage not to take the fetched instruction.

## **Interrupt Handling:**

If the interrupt signal is raised, the PC will be pushed to stack & will be updated with

0(interrupt handling routine address)

## **Control Unit:**

We have **14** main signals which are divided into:

* **ALU** Signals which are 4 signals (A3 A2 A1 A0) depending on them we tell the ALU which instruction should be executed whenever there are any R instruction.
* **IO** signals which are 2 signals (IR IW) used for input read and input write to deal with the output peripherals.
* Two **memory signals** which are (MR MW) used for memory read and memory write to deal with the memory when ever there is load or store instructions (I instruction).
* We have one signal for the **write back** which is (MTR) memory to register used whenever we want to write something from the memory back into the register file.
* **Branch** signal which indicates whether this instruction is a J type or not.
* SH which is a **shift signal** triggered whenever we have any shifting instruction such as shift left or shift right.
* RW which is **register write** signal which indicates whether we are going to write into a register or not.
* **St** & **SST** signals which are state signal and secondary state signal which are clarified above in the (32 bits Instruction manipulation part).

## **Static branch prediction:**

We assume not taken in static prediction then we will not interrupt the pipeline until we detect the branch result if not taken then won’t change anything else we will flush the fetched instructions and jump to the new instruction.

Moreover, we have chosen this because it is faster and cheaper as if we assumed it is taken, this will consume more time and hardware which will consequently consume more money and effort in designing it, so we follow make it easy as much as you can concept.