Department of Electronic & Telecommunication Engineering
University of Moratuwa

EN2031 - Fundamentals of Computer Organization and Design

PROCESSOR DESIGN PROJECT

MANIMOHAN T. *200377M*
MITHUSHAN K. *200398D*
SAIRISAN R. *200552V*

This report is submitted as assignment of module EN2031

$1^{st} March 2023$

ABSTRACT

This is Processor design project for Continuous assignment in EN2031 module.There are solutions for the questions in the Project.We have answered based on the knowledge, we got from the EN2031 Lectures which were conducted by Dr.Pasqual.

.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 DATAS FOR THE QUESTIONS

A custom processor is to be designed for a computational unit and it will be part of a System on Chip (SoC) to be placed in an embedded system for a high-speed Industry 4.0 compliant factory operation. The decision on a custom processor was made following an in-depth investigation which reveals that off-the-shelf embedded processors do not meet the throughput requirements.

The processor is not expected to run an operating system. A custom compiler will convert C language program to assembly which will be loaded to program memory. Pipeline processing is required to achieve the required throughput.

The following specifications of the processor has been given to you.

* Data Memory and Program Memory to be of size 64KB (high speed low latency) and the memory is dual port. Both Program and Data Memory can handle 16-bit words.
* A registry file R contains 8 registers : R0 to R7 where R7 acts as a Stack Pointer.
* Program Counter (PC) is considered as a special register.
* SREG is a special circular buffer that holds 8 1-bit values corresponding to results of evaluated condition in branch instructions.
* 16-bit Arithmetic and Logic Unit (ALU)
* Dedicated adders are to be used for address incrementing/decrementing where appropriate.
* Data Memory has an area reserved for a stack (limited to 4KB) and Data Memory is accessed by a separate 16-bit address bus.
* Load instructions have different variants as given below under Instruction Set details.
* The instruction set supports the following:

– Arithmetic instructions on the ALU. The supported operations are addition, subtraction, bitwise AND and bitwise OR.
– Any register in the registry file can be selected either as an operand or a result register. The selection of the one operand register and the result register must be identical. This is referred to as read-modify-write constraint.
– POP from and PUSH to the Stack. SP is decremented/incremented as part of POP/PUSH instruction using an ALU.
– Indirect register load from Data Memory. Every register in the Registry file can be specified as an address register and as a destination register. [Hint: If Data Memory is DM, Rd=DM[Ra] where Rd and Ra are any of the registers in Register File R except R7 – Stack Pointer.
– Indirect register load from Data Memory with address post-modification. Every register in the Registry file can be specified as an address register and as a destination register. In parallel with the load, the address register is incremented on the ALU (read-modify-write constraint).

- Instruction set needs to facilitate saving a 16-bit value to any register in R.
- Data transfer between any two registers in R.
- Unconditional and conditional branch instructions. The target address is specified relative to the current program counter value. It is supplied as an 8-bit signed immediate value.
- Conditional branch instructions do register comparisons for equality, less than or equal and greater than or equal conditions. The evaluated conditions are stored in SREG. [Hint: Consider using one register in the Registry file R designated as one register for branch condition comparison purposes]
- 8-bit load immediate instruction. An 8-bit signed constant is loaded to any register. 8-bit immediate value is signed extended to a 16-bit value.

You may assume reasonable values for any information that is not specifically provided above and must clearly state these assumptions. You are expected to propose appropriate opcodes for various instructions.

This is an open ended design project and marks will be given based on the approach you take in the design process. Wherever required provide appropriate justifications for your decisions.

You are required to do the following referring to the specifications given above.

2 QUESTIONS

a) .

i (20 points) List all the required instructions with proper format showing opcode and the operands and the meaning of each instruction?

* ADD Rd, Rs1, Rs2: Adds Rs1 and Rs2, and stores the result in Rd.
* SUB Rd, Rs1, Rs2: Subtracts Rs2 from Rs1, and stores the result in Rd.
* AND Rd, Rs1, Rs2 : Performs a bitwise AND operation on Rs1 and Rs2, and stores the result in Rd.
* OR Rd, Rs1, Rs2 : Performs a bitwise OR operation on Rs1 and Rs2, and stores the result in Rd.
* LOAD Rd, (Rs): Loads the 16-bit value from the memory location pointed to by Rs, and stores it in Rd.
* LOADP Rd, (Rs): Loads the 16-bit value from the memory location pointed to by Rs, and stores it in Rd. After the load, Rs is incremented by 1
* LI Rd, immediate: Loads the 8-bit immediate value.
* LUI Rd, imm[0:7]: Loads the 8-bit immediate value to upper immediate value of 16 bits Register.
* STORE (Rs), Rd: Stores the 16-bit value in Rd to the memory location pointed to by Rs
* MOV Rd, Rs: Copies the contents of Rs to Rd.
* PUSH Rs: Pushes the 16-bit value in Rs onto the stack, and decrements the Stack Pointer.
* POP Rd: Pops the top 16-bit value from the stack and stores it in Rd and increments the Stack Pointer.
* JAL imm[0:7] : Unconditionally branches to the instruction at the address given by (PC + offset).
* BEQ Rs, imm[0:7] : Branches to the instruction at the address given by (PC + offset) if the most recent comparison was for equality.
* BLT Rs, imm[0:7]: Branches to the instruction at the address given by (PC + offset) if the most recent comparison was for less than or equal.
* BGE Rs, imm[0:7]: Branches to the instruction at the address given by (PC + offset) if the most recent comparison was for greater than or equal.

ii (20 points) Develop the Instruction Formats required to support the above-mentioned instructions and provide justifications for your decisions? (Hint: You need separate formats for ALU, Control Flow, Load, Store and Register move Instructions)

ALU instructions	ADD SUB AND OR
LOAD INSTRUCTIONS	LOAD LOADP LI LUI
STORE INSTRUCTIONS	STORE
REGISTER MOVE INSTRUCTIONS	MOV
CONTROL FLOW INSTRUCTIONS	PUSH POP Jal BEQ BLT BGE

Table 1 — Instructions

* ALU instructions format:
  4 bits for opcode, 3 bits for destination register (Rd) which also acts as a one of the source register, 3 bits for source register (Rs), and 6 bits for padding. The read-modify-write constraint is achieved by forcing the destination and source registers to be the same.

opcode	Rs	Rd	func6

Table 2 — ALU instructions format

* Control Flow instructions format:
  i. Branching instructions
  4 bits for opcode, 3 bits for source register Rs, , 8 bits for storing the immediate value by which value the bit for padding. branching is supposed to be done from the current instruction.

opcode	Rs	func	Immediate [0:7]

Table 3 — Branching instructions

  ii. Stack instructions
  4 bits for the opcode, 3 bits for the source destination (Rs), 9 bits for padding.

opcode	Rs	func9

Table 4 — Stack instructions

* Load instructions format:

  i. .

  4 bits for opcode, 3 bits for destination register (Rd), 3 bits for address register (Rd), and 6 bits for padding.

opcode	Rs	Rd	func6

Table 5

  ii. load immediate (LUI):

  4 bits for opcode, 3 bits for destination register Rd, 8 bits for immediate value, 1 bit for padding.

Opcode (4)	Immediate [0:1]	func1	Rd (3)	Immediate [2:7]

Table 6 — load immediate (LUI)

* Store instructions format:
  4 bits for opcode, 3 bits for register which have address (k), 3 bits for source register (Rd), and 4 bits for padding.

opcode	Rs	Rd	func6

Table 7 — Store instructions format

* Register moves instructions format:
  4 bits for opcode, 3 bits for destination register (Rd), 3 bits for source register (Rs) and 6 bits for padding.

opcode	Rs	Rd	func6

Table 8 — Register moves instructions format

Justification:
Instruction format designed in such a way to ease the decoding process.

1 Opcode remain in same position in all instructions
2 Rs, Rd, func all are places in same position whenever it appears.
3 Last 6 bits of the immediate value also kept in same position

b) (15 points) Propose an appropriate encoding scheme for the opcodes that will faciliate easy decoding of the instructions?

Encoding scheme

instruction	opcode	func	INSTRUCTION	OPCODE	func
ADD	0000	0000	PUSH	1001	000000000
SUB	0001	0000	POP	1010	000000000
AND	0010	0000	JAL	1011	-
OR	0011	0000	BEQ	1100	-
LOAD	0100	0000	BGE	1101	-
LOADP	0101	0000	BLT	1110	-
LUI	0110	0	LI	1111	
MOV	0111	0000			
STORE	1000	0000			

Table 9 — Encoding scheme

register	code
R0	000
R1	001
R2	010
R3	011
R4	100
R5	101
R6	110

Table 10 — Register code

c) (20 points) Draw a clearly labelled datapath with all functional elements for your chosen design approach. Show all the control signals for the datapath elements?

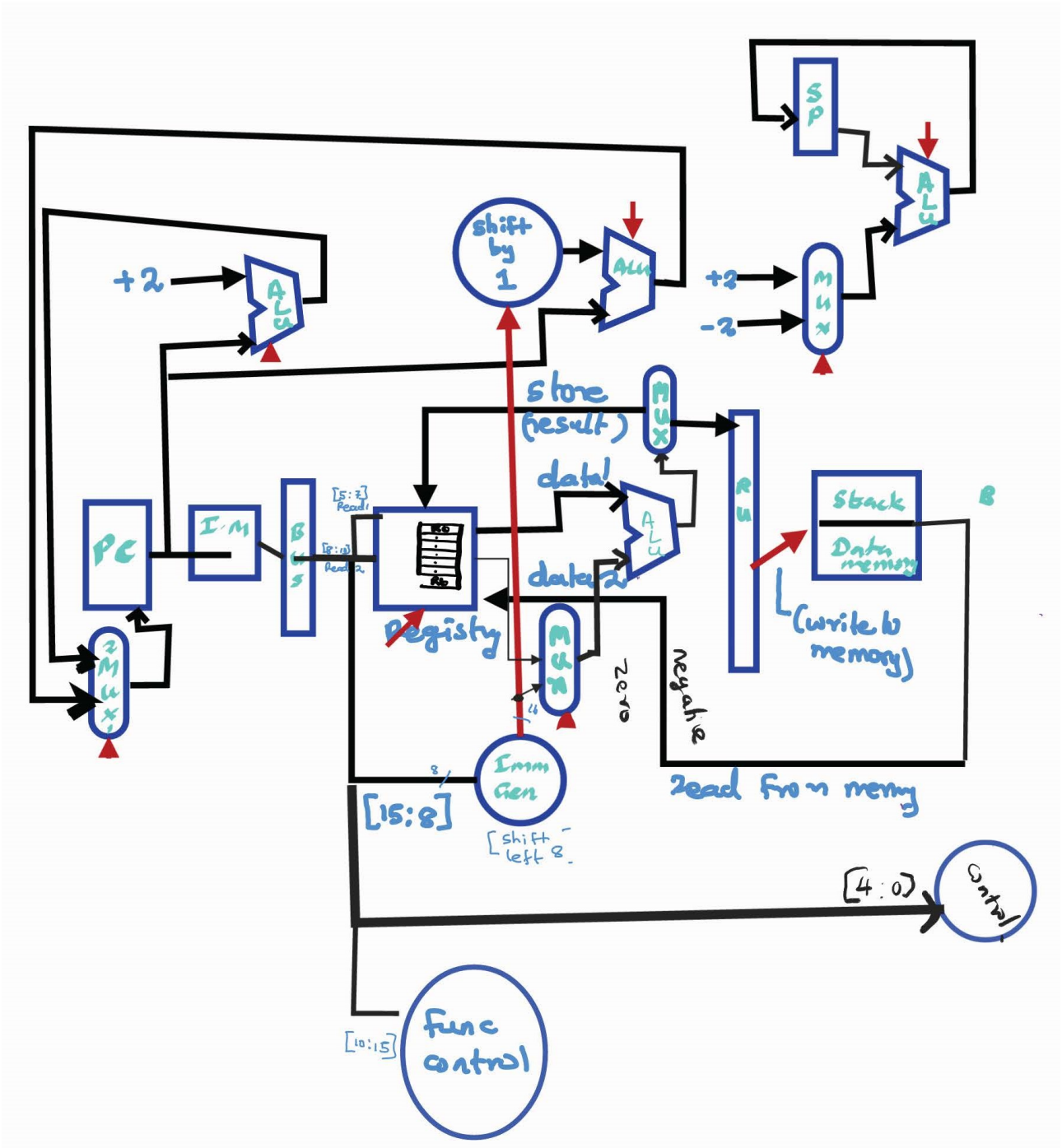

Figure 1 — Data path

d) .

i (5 points) Explain your chosen design approach for the microarchitecture: Hardwired or Microprogrammed with clear justification?
Hardwired is faster than Microprogrammed in the design.

The design is Hardwired because we have 16 instructions each mapped in to 4 bit. There is no complex operation here because of this easily decodable assignments. And

reigiters other than R7 can be used for both store and load. Since the Hardwired design can implement this for better performance.

ii (20 points) Design the controller for the datapath that you have designed above, giving reasons for your decisions. Clearly illustate the controller showing the inputs and outputs?

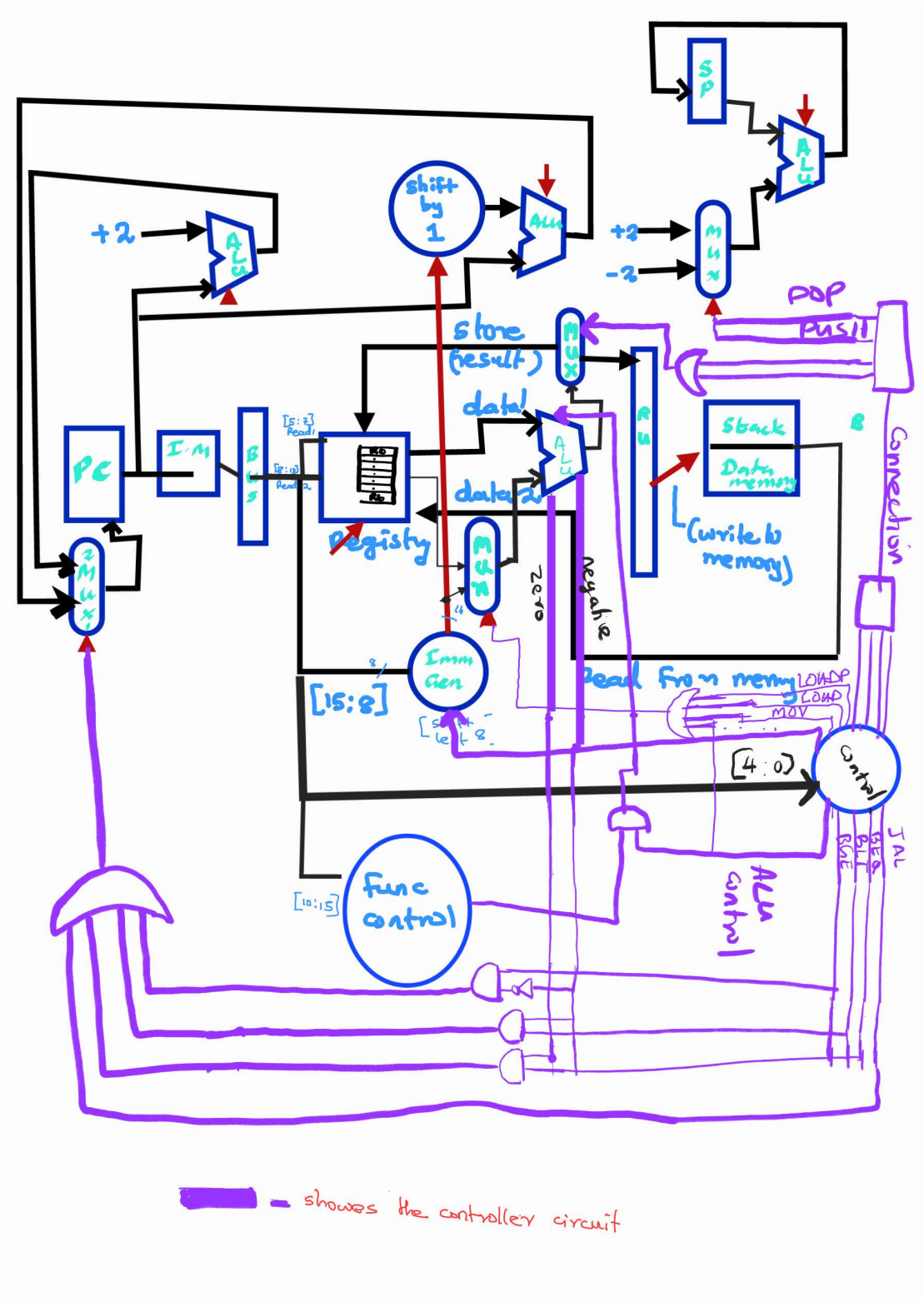

Figure 2 — Controller path