Analysis on Multi-Cycle Processor Design

제출일	2023, 06, 16	전공	컴퓨터학과
과목	컴퓨터구조	학번	2023320046
담당교수	김영근	이름	정승우

제출일	2023, 06, 16	전공	컴퓨터학과
과목	컴퓨터구조	학번	2019320076
담당교수	김영근	이름	한건희

Contents

#1 Analyze given multi-cycle ARM processor source code

Sub-modules.

1. **signextmux**
   A. This module takes a 24-bit immediate value and a 2-bit select signal, and produces a 32-bit extended immediate value. The select signal determines whether the immediate value is sign-extended or zero-extended. And depending on the select signal, different bits of 'extVal' are assigned values from the 'in' input or constant values.
   B. Inputs:
      i. [23:0] in: Input data to be extended and multiplexed.
      ii. [1:0] select: Select input to determine the operation
   C. Outputs:
      i. [31:0] extVal: Extended and multiplexed output value.

2. **register-1bit**
   A. This module implements a 1-bit register. In the rising edge of the clock or reset signal, the output is updated. Reset signal makes the regout to be zero, and if write signal is set, the regin updates the regout.
   B. Inputs:
      i. regin: Input value to be stored in the register.

ii. clk: Clock signal.

iii. write: Write control signal.

iv. reset: Reset control signal.

C. Outputs:

i. regout: Output value stored in the register.

3. register

A. This module implements a 32-bit register. In the rising edge of the clock, the output is updated. Reset signal makes the regout to be zero, and if write signal is set, the regin updates the regout.

B. Inputs:

i. [31:0] regin: Input value to be stored in the register.

ii. clk: Clock signal.

iii. write: Write control signal.

iv. reset: Reset control signal.

C. Outputs:

i. [31:0] regout: Output value stored in the register.

4. decoder_4to16

A. This module takes a 4-bit input and produces a 16-bit output. It uses a case statement to assign the correct value to 'out' based on the input 'in'.

B. Inputs:

i. [3:0] in: The input value to be decoded.

C. Outputs:

i. [15:0] output: A 16-bit value that only one bit 1 and the others are 0s.

5. registerfile

A. This module contains 16 instances of the register module, each

representing a single register. And 'reg1' and 'reg2' select the two registers.

B. Inputs:

i. [3:0] reg1: Selects the first register whose value needs to be read.

ii. [3:0] reg2: Selects the second register whose value needs to be read.

iii. [3:0] regdst: Selects the register destination for writing.

iv. [31:0] regsrc: Determines the source value to be written to the register file.

v. clk: The clock signal for register operations.

vi. reset: Signal indicating whether to reset the register file.

vii. we: It is write enable code indicating whether to update the register file with regsrc.

C. Outputs:

i. [31:0] out1: The value stored in the first selected register.

ii. [31:0] out2: The value stored in the second selected register.

iii. [31:0] pc: The value stored in register 15, which serves as the program counter.

6. signalunit

A. This module represents a control unit that generates control signals based on the current state of the processor.

B. Inputs:

i. clk: Clock signal for the module.

ii. reset: Reset signal for the module.

iii. [11:0] flags: Flags representing the state of the processor.

iv. zero: Signal indicating whether the result of the previous operation was zero.

C. Outputs:

i. Mwrite: Signal indicating whether to enable memory write operation.

ii. IRwrite: Signal indicating whether to enable instruction register write

operation.

iii. Mread: Signal indicating whether to enable memory read operation.

iv. regwrite: Signal indicating whether to enable register write operation.

v. [1:0] regdst: Selects the destination register for writing.

vi. [1:0] regsrc: Selects the source register for reading.

vii. [1:0] ALUsrcA: Selects the source for operand A in the ALU operation.

viii. [1:0] ALUsrcB: Selects the source for operand B in the ALU operation.

ix. [3:0] ALUop: Represents the ALU operation to be performed.

x. NZCVwrite: Signal indicating whether to enable flags (NZCV) write operation.

xi. [1:0] immsrc: Selects the source for immediate value in the instruction.

xii. regbdst: Signal indicating whether the destination of the branch is a register.

7. ALUopdecoder

A. This module decodes the instruction opcode (instop) to determine the ALU operation (aluop) to be performed.

B. Inputs:

i. [3:0] instop: The instruction opcode for decoding the ALU operation.

C. Outputs:

i. [2:0] aluop: The decoded ALU operation based on the instruction opcode. It selects the appropriate logic or arithmetic operation in the ALU32bit module.

8. ALU32bit

A. This module represents a 32-bit Arithmetic Logic Unit (ALU) that performs arithmetic and logical operations on two 32-bit inputs.

B. Inputs:

i. [31:0] inpa: Input operand A for the ALU operation.

ii. [31:0] inpb: Input operand B for the ALU operation.

iii. cin: Carry-in signal for arithmetic operations.

iv. [2:0] aluop: Determines the ALU operation to be performed.

C. Outputs:

i. [31:0] result: The result of the ALU operation.

ii. negative: Signal indicating whether the result is negative.

iii. zero: Signal indicating whether the result is zero.

iv. cout: Carry-out signal for arithmetic operations.

v. overflow: Signal indicating whether overflow occurred during the operation.

'armreduced' module

- **Inputs:**
  - clk: Clock signal used for synchronous operations.
  - reset: Reset signal that initializes the processor.
  - inst: Input instruction to be executed.
  - nIRQ: Input signal indicating whether an interrupt request is received.
- **outputs:**
  - pc: Program counter, representing the current instruction address.
  - be: Output indicating which bytes of a word should be enabled during a memory access.
  - memaddr: Memory address for memory operations.
  - memwrite: Signal indicating a memory write operation.
  - memread: Signal indicating a memory read operation.
  - writedata: Data to be written to memory.
  - readdata: Data read from memory.

- **Internal signals and registers**
  - IRwrite: Signal indicating whether the instruction register (IR) should be written.
  - regwrite: Signal indicating whether a register write operation should be performed.
  - NZCVwrite: Signal indicating whether the flags register should be written.
  - regdst: Destination register for register write operations.
  - regsrc: Source register for register write operations.
  - ALUsrcA: ALU source selection for operand A.
  - ALUsrcB: ALU source selection for operand B.
  - ALUop: ALU operation to be performed.
  - instop: Decoded ALU operation based on the instruction.
  - regBdst: Destination register for the second source operand of ALU operations.
  - imm: Immediate value extracted from the instruction.

Instances of the above submodules

MDR: This submodule is an instance of the register module. It stores the data read from memory (readdata) and provides it as an output (mdr) when required.

ALUoutRegister: This submodule is another instance of the register module. It stores the result of the ALU operation (ALUresult) and provides it as an output (ALUout) when needed.

A_Register and B_Register: These submodules are instances of the register module. They store the values of registers A and B, respectively, and provide them as outputs (A and B) for subsequent operations.

InstructionRegister: This submodule is also an instance of the register module. It stores the instruction (inst) and provides it as an output (instructions) for decoding and execution.

Z and C registers: These submodules are instances of the register_1bit module. They store the values of the zero flag (z) and the carry flag (c), respectively, and provide them as outputs for the NZCV register.

SignalControl: This submodule is responsible for controlling the signals that control the behavior of the ARM processor. It takes inputs from the instruction (instructions), zero flag (z), and other control signals, and generates outputs that control the behavior of other submodules.

RegisterFile: This submodule is an instance of the registerfile module. It serves as the register file for the ARM processor, storing and providing the values of registers based on the control signals (reg1, reg2, regdst, regsrc).

Immidiate: This submodule is an instance of the signextmux module. It handles sign extension and selection of immediate values (instructions[23:0]) based on the control signal (immsrc), providing the extended immediate value (imm) as an output.

ALUopDecoder: This submodule decodes the instruction operation (instop) and provides the corresponding ALU operation (ALUop) as an output.

ALU: This submodule is an instance of the ALU32bit module. It performs the arithmetic and logical operations specified by the ALUop control signal on the inputs (ALUnum1, ALUnum2) and provides the result (ALUresult) and other flags (ALUflags) as outputs.

Operation process of the ‘armreduced’ module

1. Instruction Fetch:

The 'InstructionRegister' submodule receives the 'inst' input signal, which represents the fetched instruction.
The instruction is stored in the 'InstructionRegister' and is output as 'instructions'.

2. Register Fetch:

The 'SignalControl' submodule takes the 'instructions' signal as input and generates various control signals required for instruction execution.

Control signals include 'IRwrite' for writing to the instruction register, 'memwrite' for enabling memory write, 'memread' for enabling memory read, 'regwrite' for enabling register write, 'regdst' for selecting the destination register, 'regsrc' for selecting the source register, 'ALUsrcA' for selecting the first ALU operand, 'ALUsrcB' for selecting the second ALU operand, 'ALUop' for selecting the ALU operation, 'NZCVwrite' for writing the condition flags, 'immsrc' for selecting the source of immediate value, and 'regBdst' for selecting the second source register.

The 'RegisterFile' submodule takes the control signals and other inputs to perform register file operations. It reads the values from the specified source registers and outputs them as 'readA' and 'readB'. It also selects the destination register based on the 'regdst' signal and outputs it as 'RFdst'. The values to be written to the register file are selected based on the 'regsrc' signal and output as 'RFsrc'. The 'RegisterFile' module also contains the program counter (PC) register, which is output as 'pc'.

3. Execution

The 'ALUopDecoder' submodule takes the 'instop' signal from the control unit and decodes it into the appropriate ALU operation, which is output as 'ALUop'.
The 'ALU32bit' submodule performs arithmetic and logical operations based on the selected ALU operation, operands, and control signals.
It takes the first operand from 'ALUnum1' based on the 'ALUsrcA' signal and the second operand from 'ALUnum2' based on the 'ALUsrcB' signal.
The ALU32bit module performs the ALU operation and outputs the result as 'ALUresult'.
It also generates the necessary condition flags, including 'negative', 'zero', 'carry', and 'overflow', which are output as 'ALUflags'.

4. Memory:

The 'memaddr' output from the 'armreduced' module is the memory address to be accessed.
The 'memwrite' and 'memread' signals determine whether a memory write or read operation should be performed.
The 'writedata' signal contains the data to be written to memory.
The 'MDR' submodule acts as a memory data register and stores the data read from memory, which is output as 'mdr'.

5. Write Back:

The result of the previous stage, ALUresult, is written back to the destination register specified in the instruction.The result is stored in the appropriate register in the RegisterFile. Additionally, the flags register (NZCV) is updated based on the ALU operation.

#2 Analyze Signal unit

Signalunit module is used in ARMCPU.v file, and its inputs and outputs are these :

Input :

Variables	Size (bit)	Description

clk, reset	1bit	Clock & Reset signals
zero	1bit	Condition Flag : Zero
flags	12bit	Top 12 bits of instructions to make control signals

Output :

Variables	Size (bit)	Description
Mwrite	1	Write Enable signal of Memory
IRwrite	1	Write Enable signal of Instruction Register
Mread	1	Read Enable signal of Memory
regwrite	1	Write Enable signal of Registers
Regdst	2	Register destination of Regiter Writing operation 00 : Rd(2nd operand) 01 : R15 – PC 10 : R14 – LR
Regsrc	2	Value of Register Writing operation 00 : MDR 01 : ALUOUT 10 : ALUresult 11 : B (second operand of ALU)
ALUsrcA	2	Source of Register A (first operand of ALU) 00 : PC 01 : A 10 : zero
ALUsrcB	2	Source of Register B (second operand of ALU) 00 : +4 01 : +8 10 : imm 11 : B shift shmt
ALUop	4	The operation of ALU 0000 : and 0001 : xor 0010 : sub 0100 : add 0101 : adc

		0110 : sbc 1010 : sub 1100 : or 1101 : add
NZCVwrite	1	Write Enable signal of NZCV flags
Immsrc	2	Source of Immediate Value 00 : 8_signext 01 : 12_zeroext 10 : 24_signext
regbdst	1	Select Signal of regBread MUX 0 : inst[3:0] 1 : inst[15:12]

In signalunit module, the wire s is 2nd dimensional array.

For each row, the signals are saved as a array of 0s and 1s.

The wire total contains the total steps to execute the instruction, and the wire step contains the current step for current clk cycle.

oneAdder Step module adds 1 to current step value, and stops when step == total.

Since, for every instructions, the operation for first and second step is same, the s[0] and s[1] value is assigned regardless of the flags. (upper 12 bits of instructions)

step	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x

The signalcontrol bringSignal module derives the signals by decoding the upper 12 bits

of instructions. Its inputs and outputs are these :

Input :

Variables	Size (bit)	Description
zero	1bit	Condition Flag : Zero
flags	12bit	Top 12 bits of instructions to make control signals

Output :

Variables	Size (bit)	Description
total	3	Total steps required to execute the instruction.
s2	20	s[2] value
s3	20	s[3] value
s4	20	s[4] value

In signalcontrol module, the value of the first if statement is :

flags[11]&flags[10]&flags[9])||(flags[8]^zero)

The first operand of or operator detects whether flags[11:9] == 3b'111, which refers Always condition code 1110 (we can ignore 1111 condition code since it is not an usual case). The second operand of or operator detects two possible cases :

If zero value is 1, the flags[8] should be 0.

If zero value is 0, the flags[8] should be 1.

If the if statement is not taken, the module generates the signals for recovery instruction.

The rest of bringsignal will be decoded in my python code.

(https://github.com/we4t/COSE222_CA/blob/main/2nd_Project/TermProject_Multicycle/controlsig_analyze.py)

```python
def get_instructions(flags, zero):
    if flags[0:3] == "111" or (zero == 1 and flags[3] == "0" or flags[3] == "1" and zero == 0):
        if flags[4] == "1":  # B, BL
            if flags[7] == "0":  # B
                return [
                    "B",
                    3,
                    "01110110000101000xxx",
                    "0000xxxx00000100xxx",
                    "00010110001001000100",
                ]

            else:  # BL
                return [
                    "BL",
                    4,
                    "01110110000101000xxx",
                    "0000xxxx00000100xxx",
                    "00011001001001000100",
                    "00010101xxxxxxxx0xxx",
                ]

        elif flags[5] == "1":  # LDR, STR
            if flags[11] == "0":
                # STR
                # Load/store immediate offset   NZCV010PUBW0
                # Load/store register offset    NZCV011PUBW0
                # ex_flags                      11100101100

                flag5 = "00"
                if flags[6] == "1":
                    flag5 = "11"
                else:
                    flag5 = "10"
                # decode immediate / register offest

                flag3 = "0100" if flags[8] == "1" else "0010"
                # decode whether offset is added from base (U == 1) or subtracted from base (U == 0)
                flag0 = "1"
                # decode the destination of regbdst

                return [
                    "STR",
                    4,
                    "01110110000101000xxx",
                    "0000xxxx00000100xxx",
                    "0001010101" + flag5 + flag3 + "001" + flag0,
                    "1000xxxxxxxxxxxx0xxx",
                ]

            else:
                # LDR
                # Load/store immediate offset   NZCV010PUBW1
                # Load/store register offset    NZCV011PUBW1
                # ex_flags                      11100101100


                flag5 = "11" if flags[6] == "1" else "10"
                # decode immediate / register offest
                flag3 = "0100" if flags[8] == "1" else "0010"
                # decode whether offset is added from base (U == 1) or subtracted from base (U == 0)
                flag0 = "0"
                # decode the destination of regbdst

                return [
                    "LDR",
                    5,
                    "01110110000101000xxx",
                    "0000xxxx00000100xxx",
                    "0001010101" + flag5 + flag3 + "001" + flag0,
                    "0010xxxxxxxxxxx0xxx",
                    "00010000xxxxxxxx0xxx"
                ]

        elif flags[7:11] == "1010":
            # CMP
            # Data processing immediate shift   :   NZCV000opcdS
            # Data processing immediate         :   NZCV001opcdS
            # ex_flags =                            11100110101


            flag5 = "10" if flags[6] == "1" else "11"
            # decode whether shift or not

            return [
                "CMP",
                3,
                "01110110000101000xxx",
                "0000xxxx00000100xxx",
                "0001010101" + flag5 + "00101000",
            ]

        elif flags[7:11] == "1101":
            # MOV
            # Data processing immediate shift   :   NZCV000opcdS
            # Data processing immediate         :   NZCV001opcdS
            # ex_flags =                            11100111010


            flag5 = "10" if flags[6] == "1" else "11"
            # decode whether shift or not
            flag0 = flags[11]
            # set flag, update NZCV

            return [
                "MOV",
                4,
                "01110110000101000xxx",
                "0000xxxx00000100xxx",
                "0001010110" + flag5 + "0100" + flag0 + "000",
                "00010001xxxxxxxx0xxx",
            ]

        else:
            # ALU
            # Data processing immediate shift   :   NZCV000opcdS
            # Data processing immediate         :   NZCV001opcdS
            # ex_flags =                            11100001001


            flag5 = "10" if flags[6] == "1" else "11"
            # decode whether shift or not
            flags_str = flags[7:]
            # opcode + NZCV update flag

            return [
                "ALU",
                4,
                "01110110000101000xxx",
                "0000xxxx00000100xxx",
                "0001010101" + flag5 + flags_str + "000",
                "00010001xxxxxxxx0xxx",
            ]

    else:
        # Recovery
        return [
            "Recovery",
            3,
            "01110110000101000xxx",
            "0000xxxx00000100xxx",
            "00010101xxxxxxxx0xxx",
        ]
```

For every instructions, the following S value is written like below :

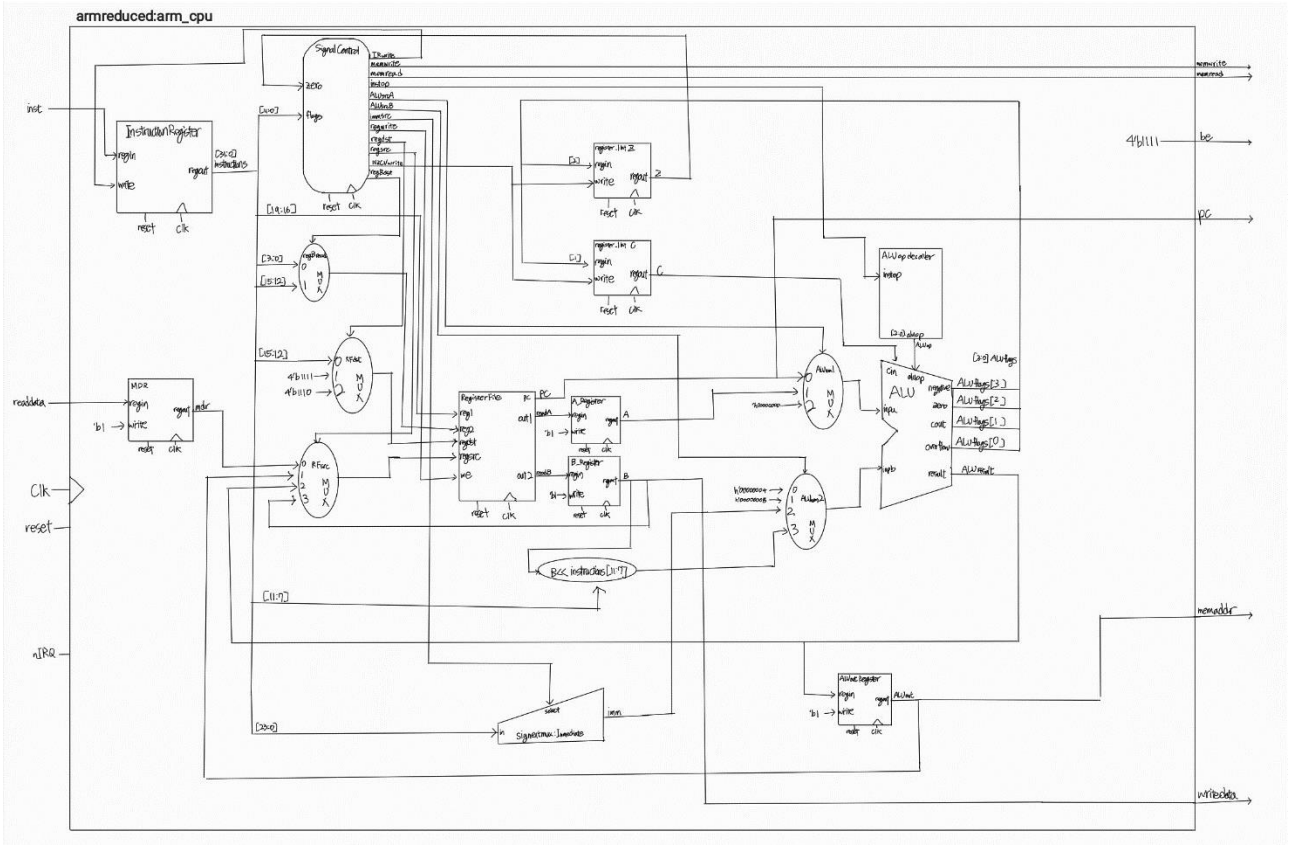
(There might be some differences between immediate offset, register offset for STR / LDR instructions, and Data processing immediate shift and Data processing immediate for ALU operations. For STR / LDR instructions, the signals are generated based on register offset. For Data processing instructions, the signals are generated based on Data processing immediate. (no shift) The difference is noted in python code above, and the flags value for generating the signals are written also)

B	M write	IR write	Mread	reg write	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	10 ALUresult	00 PC	10 imm	0100 add	0	10 24_signext	0 Rm (third Operand)
3	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
BL	M write	IR write	Mread	reg write	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	10 R14 - LR	01 ALUOUT	00 PC	10 imm	0100 add	0	10 24_signext	0 Rm (third Operand)
3	0	0	0	1	01 R15 - PC	01 ALUOUT	xx	xx	xxxx	0	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
STR	M write	IR write	Mread	reg write	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	01 A	10 imm	0100 add	0	01 12_zeroext	1 Rd (second Operand)
3	1	0	0	0	xx	xx	xx	xx	xxxx	0	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x

LDR	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	01 A	10 imm	0100 add	0	01 12_zeroext	0 Rm (third Operand)
3	0	0	1	0	xx	xx	xx	xx	xxxx	0	xx	x
4	0	0	0	1	00 Rd(2nd Operand)	00 MDR	xx	xx	xxxx	0	xx	x
CMP	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	01 A	10 imm	0010 sub	1	00 8_signext	0 Rm (third Operand)
3	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
MOV	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	10 zero	10 imm	0100 add	0	00 8_signext	0 Rm (third Operand)
3	0	0	0	1	00 Rd(2nd Operand)	01 ALUOUT	xx	xx	xxxx	0	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
ALU	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	01 A	10 imm	1101 add	0	00 8_signext	0 Rm (third Operand)
3	0	0	0	1	00 Rd(2nd Operand)	01 ALUOUT	xx	xx	xxxx	0	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x

Recovery	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	xx	xx	xxxx	0	xx	x
3	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x
4	x	x	x	x	xx	xx	xx	xx	xxxx	x	xx	x

#3 Draw block diagram of multi-cycle ARM processor



#4 Explain How the Test program works on the given processor

1. MOV r1, #0 | E3A01000:

Binary code : 1110 001 1101 0 0000 0001 0000 00000000

cond : always

funct3 : Data processing immediate

opcode : MOV Move

Rd := shifter_operand (no first operand)

s: 0

Rn: 0

Rd: 1

rotate: 0

imm8: 0

MOV	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	01 00 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	00 10 sub	0	xx	x

2	0	0	0	1	01 R15 - PC	01 ALU OUT	10 zero	10 imm	0100 add	0	008_signext	0 Rm (third Operand)
3	0	0	0	1	00 Rd(2nd Operand)	01 ALU OUT	xx	xx	xxxx	0	xx	x

Initial pc : x

1) PC = PC + 8 = x + 8

2) ALUresult = PC - 4 = x + 4

3) PC = ALUout = previous ALUresult = x + 4

ALUresult = (8bit_sign-extension)(imm32) = 1

4) R[Rd] = ALUout = previous ALUresult = 1

2. ADD r1, r1, #1 | E2811001 :

Binay code : 1110 001 0100 0 0001 0001 0000 00000001

cond : always

funct3 : Data processing immediate

opcode : ADD Add

Rd := Rn + shifter_operan

s: 0

Rn: 1

Rd: 1

rotate: 0

imm8: 1

ALU	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	01 A	10 imm	1101 add	0	008_signext	0 Rm (third Operand)
3	0	0	0	1	00 Rd(2nd Operand)	01 ALUOUT	xx	xx	xxxx	0	xx	x

Initial pc : x

1) PC = PC + 8 = x + 4

2) ALUResult = PC - 4 = x + 4

3) PC = ALUOut = PC - 4 = x + 4

ALUResult = A + imm

4) R[Rd] = ALUOUT = A+imm

3. LDR r1, [r0, #0xC] | 0C1090E5:

Binary code : 1110 010 11001 0000 0001 000000001100

cond: always

funct3: Load/store immediate offset

pubwl: 11001

pubwl: Load Instruction

Rn: 0

Rd: 1

imm12: 12

LDR	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	01 00 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	00 10 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	01 A	10 imm	01 00 add	0	01 12_zeroext	0 Rm (third Operand)
3	0	0	1	0	xx	xx	xx	xx	xxxx	0	xx	x
4	0	0	0	1	00 Rd(2nd Operand)	00 MDR	xx	xx	xxxx	0	xx	x

Initial PC : x

1) PC = PC + 8 = x + 8

2) ALUresult = PC – 4 = x + 4

3) PC = ALUOut = PC-4 = x + 4

ALUresult = A + imm12_zeroext

4) Mdr = MEM[ALUOut]

5) R[Rd] = MDR

4. **BEQ reset**

Binary code : 0000 101 00000 xxx...

cond: EQ (zero flag == 1)

If current zero flag is zero, the if statement in signalcontrol module will taken and generate the branch instruction control signals. Else if current zero flag is not zero, the if statement in signalcontrol module will not taken and generate the recovery instruction control signals

For Branch Instructions :

B	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	10 ALUresult	00 PC	10 imm	0100 add	0	1024_signext	0 Rm (third Operand)

Initial PC : x

1) PC = ALUresult = PC + 8 = x + 8

2) ALUresult = PC – 4 = x + 4

3) PC = ALUresult = PC + 8 + imm(24bits, sign extension) = ...

For Recovery Instructions :

Recovery	Mwrite	IRwrite	Mread	regwrite	regdst	regsrc	ALUsrcA	ALUsrcB	ALUop	NZCVwrite	immsrc	regbdst
0	0	1	1	1	01 R15 - PC	10 ALUresult	00 PC	01 +8	0100 add	0	xx	x
1	0	0	0	0	xx	xx	00 PC	00 +4	0010 sub	0	xx	x
2	0	0	0	1	01 R15 - PC	01 ALUOUT	xx	xx	xxxx	0	xx	x

1) PC = ALUresult = PC + 8 = x + 8

2) ALUresult = PC – 4 = x + 4

3) PC = ALUOUT = x + 4 (next instruction)