Samuel Lichtenheld

10/29/15

Assignment 3

Introduction

This assignment covered the single cycle mips architecture. The basic datapath was implemented, preparing us to implement a pipeline later.

Instruction Functionality

|  |  |  |
| --- | --- | --- |
| Name | Operation | Type |
| Add | R[rd] = R[rs] + R[rt] | R |
| add imm. | R[rt] = R[rs] + SignExtImm | I |
| add imm. uns. | R[rt] = R[rs] + ZeroExtImm | I |
| add uns. | R[rd] = R[rs] + R[rt] | R |
| and | R[rd] = R[rs] & R[rt] | R |
| and imm. | R[rt] = R[rs] & ZeroExtImm | I |
| branch eq | if (R[rs]==R[rt])      PC=PC+4+BranchAddr | I |
| branch not eq | if (R[rs]!=R[rt])      PC=PC+4+BranchAddr | I |
| jump | PC=JumpAddr | J |
| jump and link | R[31]=PC+8;PC=JumpAddr | J |
| jump reg | PC=R[rs] | R |
| load byte uns. | R[rt] = {24’b0,M[R[rs]+SignExtImm](7:0)} | I |
| load halfword uns. | R[rt] = {16’b0,M[R[rs]+SignExtImm](15:0)} | I |
| load upper imm. | R[rt] = {imm, 16’b0} | I |
| load word | R[rt] = M[R[rs]+SignExtImm] | I |
| Nor | R[rd] = ~(R[rs] | R[rt]) | R |
| Or | R[rd] = R[rs] | R[rt] | R |
| Or imm. | R[rt] = R[rs] | ZeroExtImm | I |
| set less than | R[rd] = (R[rs]<R[rt]) ? 1:0 | R |
| set less than imm. | R[rt] = (R[rs]<SignExtImm) ? 1:0 | I |
| set less than imm. uns. | R[rt] = (R[rs]<SignExtImm) ? 1:0 | I |
| set less than uns. | R[rd] = (R[rs]<R[rt]) ? 1:0 | R |
| shift left logical | R[rd] = R[rt] << shamt | R |
| shift right logical | R[rd] = R[rt] >>> shamt | R |
| store byte | M[R[rs]+SignExtImm}(7:0) = R[rt](7:0) | I |
| store halfword | M[R[rs]+SignExtImm}(15:0) = R[rt](15:0) | I |
| store word | M[R[rs]+SignExtImm} = R[rt] | I |
| subtract | R[rd] = R[rs] -R[rt] | R |
| subtract uns. | R[rd] = R[rs]= R[rt] | R |

Implentation Order

# Set1

All R-type instructions except jr

add

addi

addu

and

or

nor

slt

sltu

sll

srl

sub

subu

# Set2

beq

bne

jal

jr

j

**Set3**

lbu

lhu

lw

sb

sh

sw

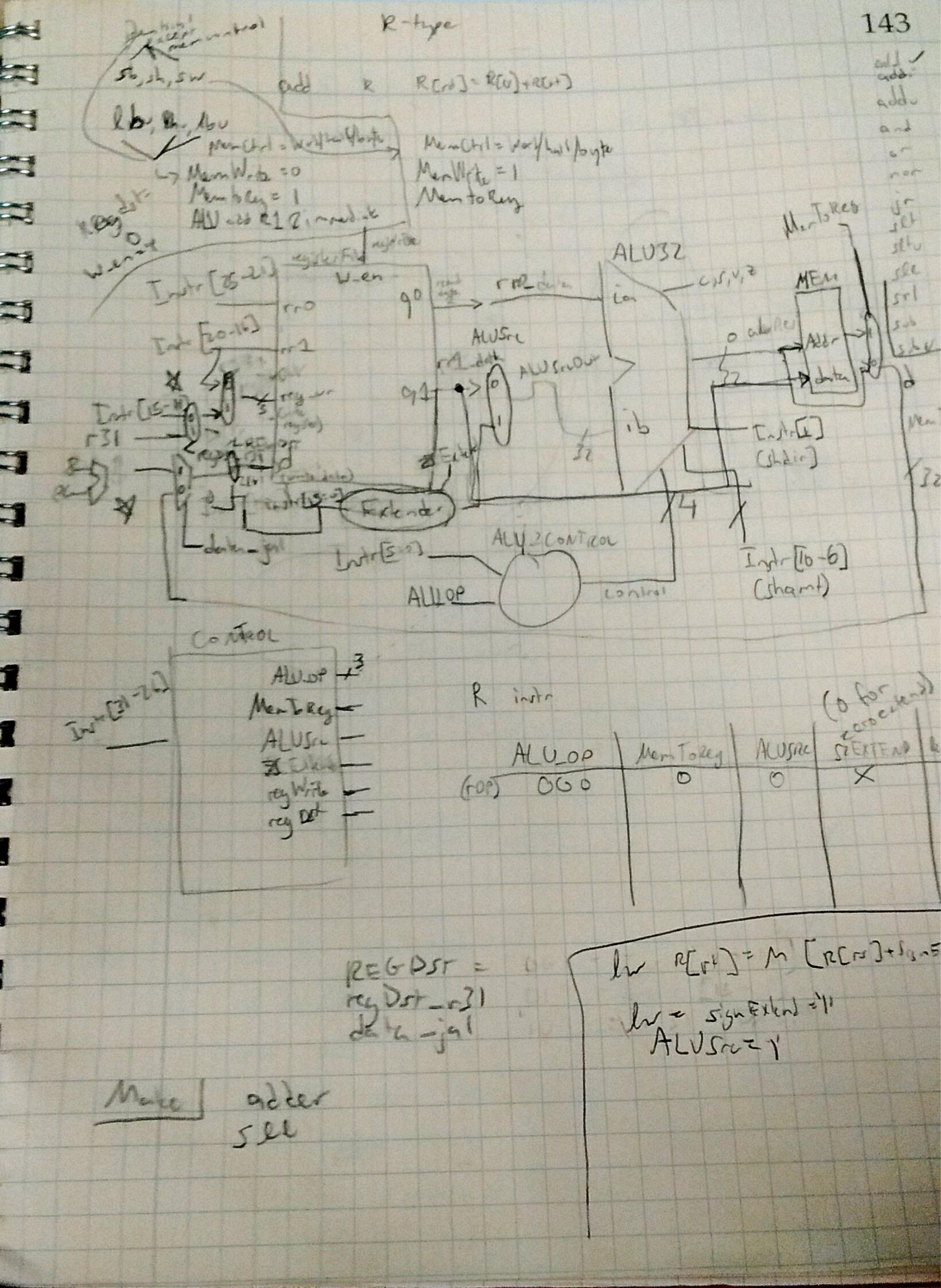


Figure : Everything but Load and Stores without PC

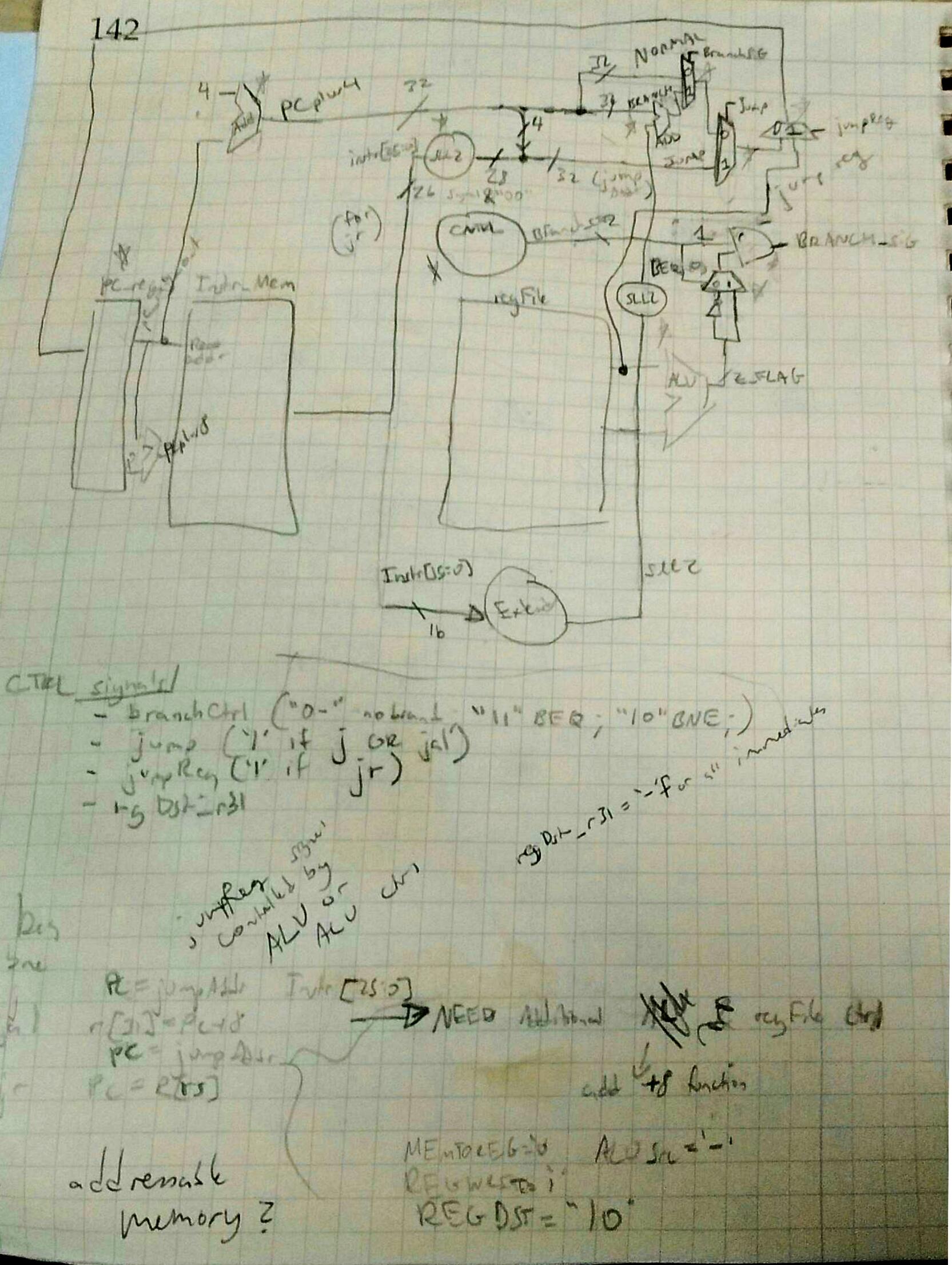


Figure : Everything PC related (jumps and branches)

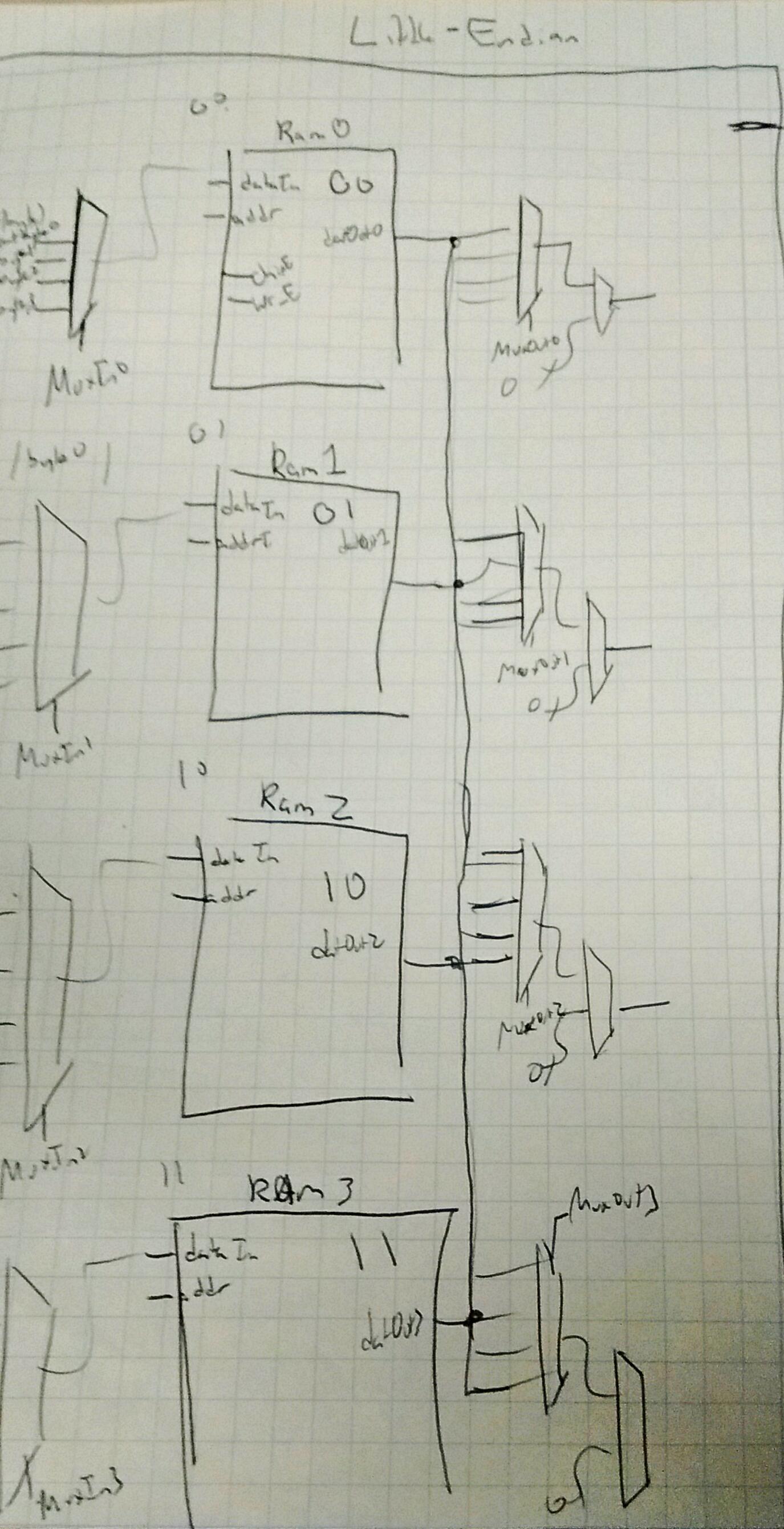


Figure : Byte Addressable Mem

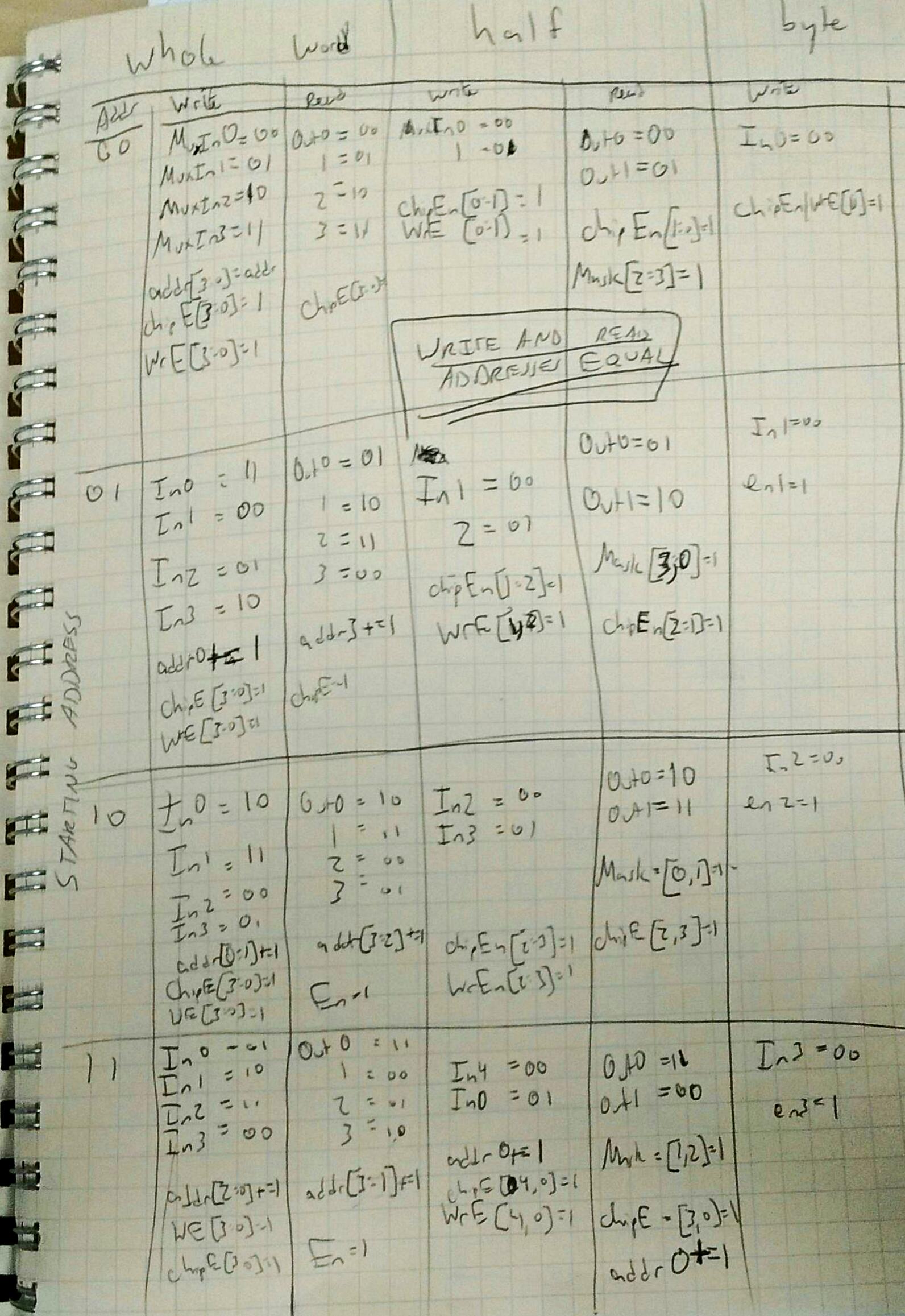


Figure : Byte Addressable Mem Control Signals (Little Endian)

# ROP\_TEST

addi $1, $0, 0x0001

addi $2, $0, 0xffff

add $3, $2, $1

addu $3, $2, $1

sub $3, $2, $1

sub $3, $1, $2

lui $1, 0xffff

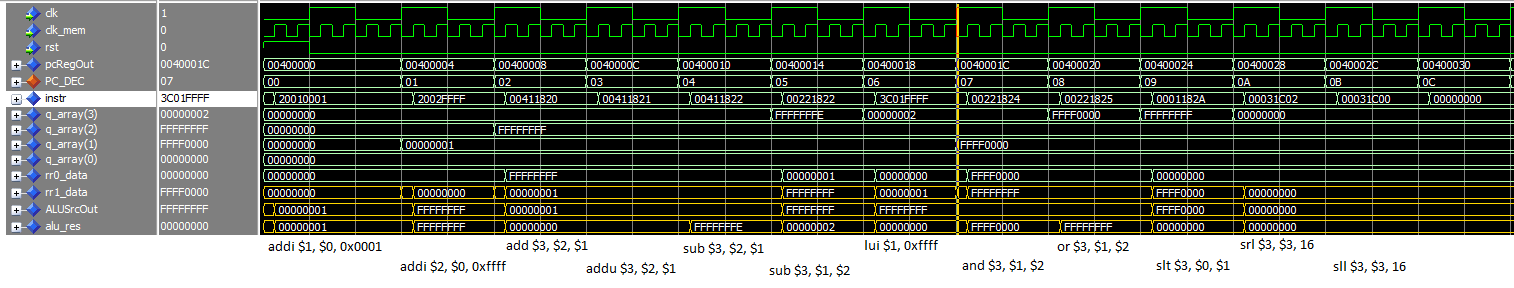
and $3, $1, $2

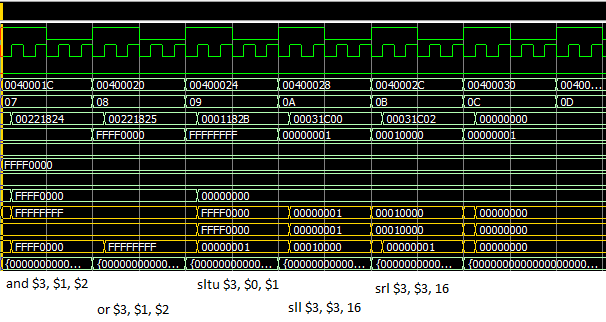
or $3, $1, $2

sltu $3, $0, $1

sll $3, $3, 16

srl $3, $3, 16





Branch & Jump testing through Fibonacci Sequence

0: addiu $1, $0, 0x0005

1: add $4, $0, $0

2: add $2, $r0, $0

3: addi $5, $0, 0x0001

for:

4: slt $10, $2, $1

5: beq $10, $0, exit(9)

6: slti $10, $2, 0x0002

7: beq $10, $0, else(2)

8: add $3, $2, $0

9: j forloopend

else:

a: add $3, $4, $5

b: add $4, $5, $0

c: add $5, $3, $0

forloopend:

d: addi $2, $2, 0x0001

e: j for

exit:

f: j exit

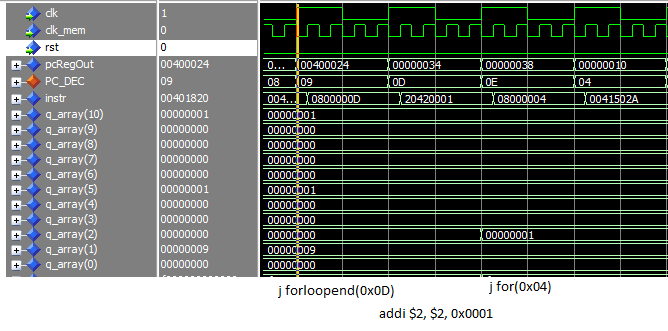


Figure : jumps exhibited

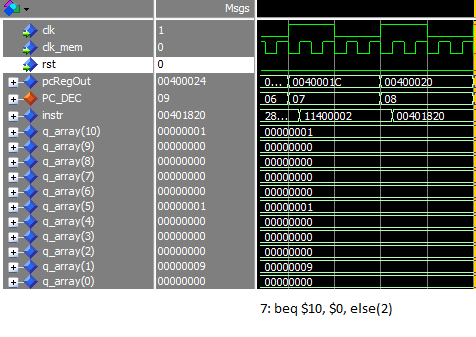


Figure : BEQ if equality isn't satisfied

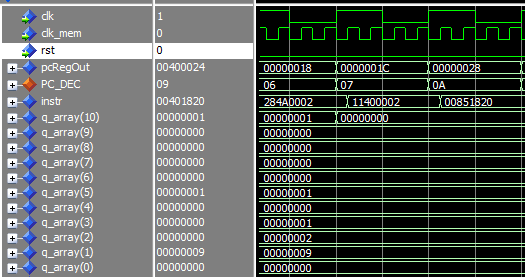


Figure : BEQ if equality is satisfied

Byte Addressable Ram Testing

0: lui $1, 0x0100

1: lui $2, 0xFFFF

2: lui $3, 0x5555

3: lui $4, 0x8888

4: addi $2, 0xFFFF

5: addi $3, 0x5500

6: addi $4, 0x0088

7: addi $5, 0xDDDD

8: addi $6, 0x0001

9: addi $7, 0x0012

a: addi $8, 0x0023

b: sw $1, 0($0)

c: sw $2, 4($0)

d: sw $3, 8($0)

e: sw $4, 12($0)

f: lw $20, 12($0)

10: lw $21, 8($0)

11: lw $22, 4($0)

12: lw $23, 0($0)

13: sw $1, 0($6)

14: sw $2, 4($6)

15: sw $3, 8($6)

16: sw $4, 12($6)

17: lw $20, 12($6)

18: lw $21, 8($6)

19: lw $22, 4($6)

1a: lw $23, 0($6)

….Continued with various test cases on all edges

Waveforms on next page

