Lab 6: PUnC - A Microprocessor
Final

Daniel Simone

14 December 2021

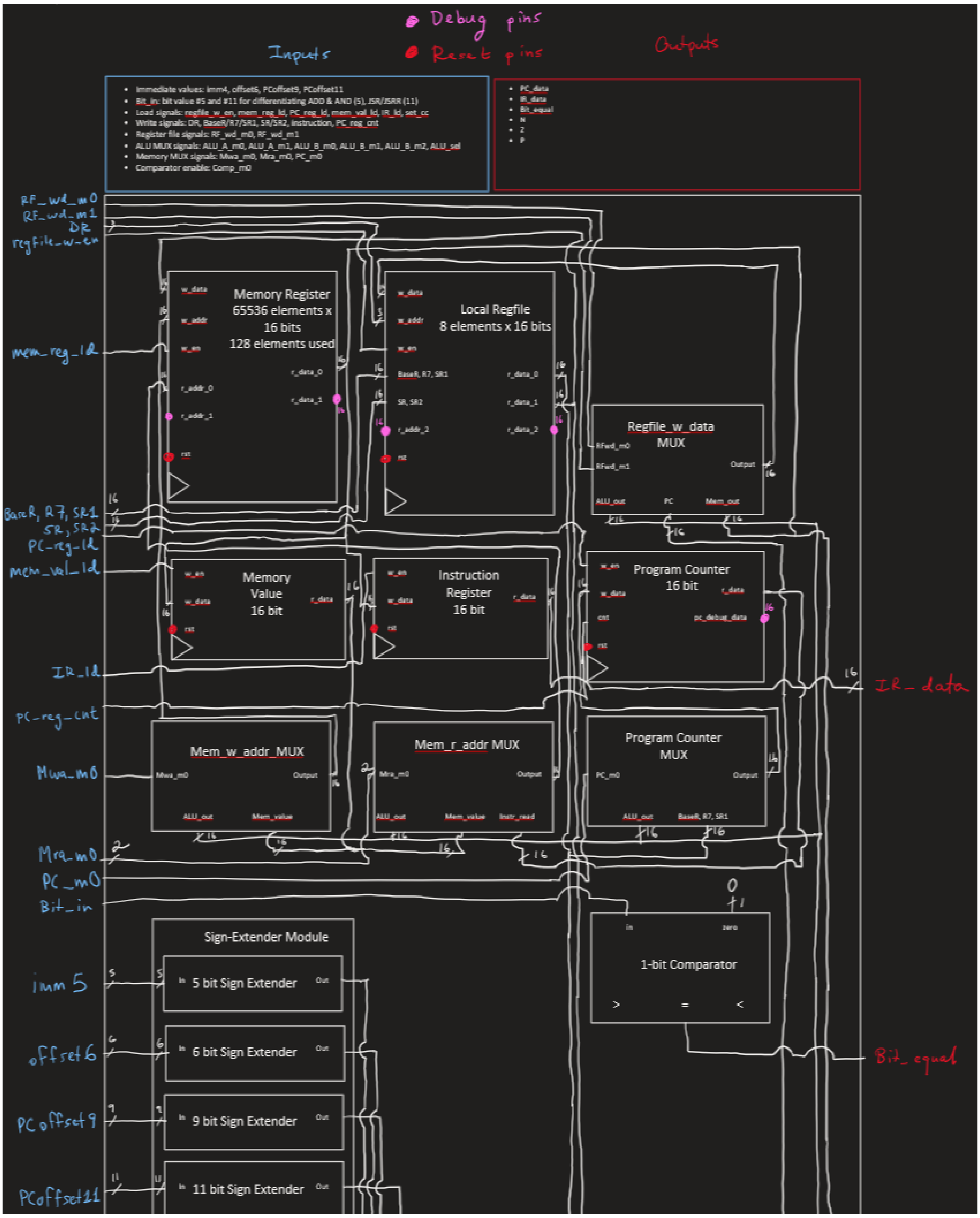
1 Introduction

1.1 Class Information

Group: Daniel Simone & Christian (Hugh) Skinker
Course: ECE 206
Demo Lab section: B02
Lecture Instructor: Professor Sharad Malik

2 Write-Up

2.1 Revised Datapath


Inputs

- Immediate values: imm4, offset6, PCoffset9, PCoffset11
- Bit_in: bit value #5 and #11 for differentiating ADD & AND (5), JSR/JSRR (11)
- Load signals: regfile_w_en, mem_reg_ld, PC_reg_ld, mem_val_ld, IR_ld, set_cc
- Write signals: DR, BaseR/R7/SR1, SR/SR2, instruction, PC_reg_cnt
- Register file signals: RF_wd_m0, RF_wd_m1
- ALU MUX signals: ALU_A_m0, ALU_A_m1, ALU_B_m0, ALU_B_m1, ALU_B_m2, ALU_sel
- Memory MUX signals: Mwa_m0, Mra_m0, PC_m0
- Comparator enable: Comp_m0

Outputs

- PC_data
- IR_data
- Bit_equal
- N
- Z
- P

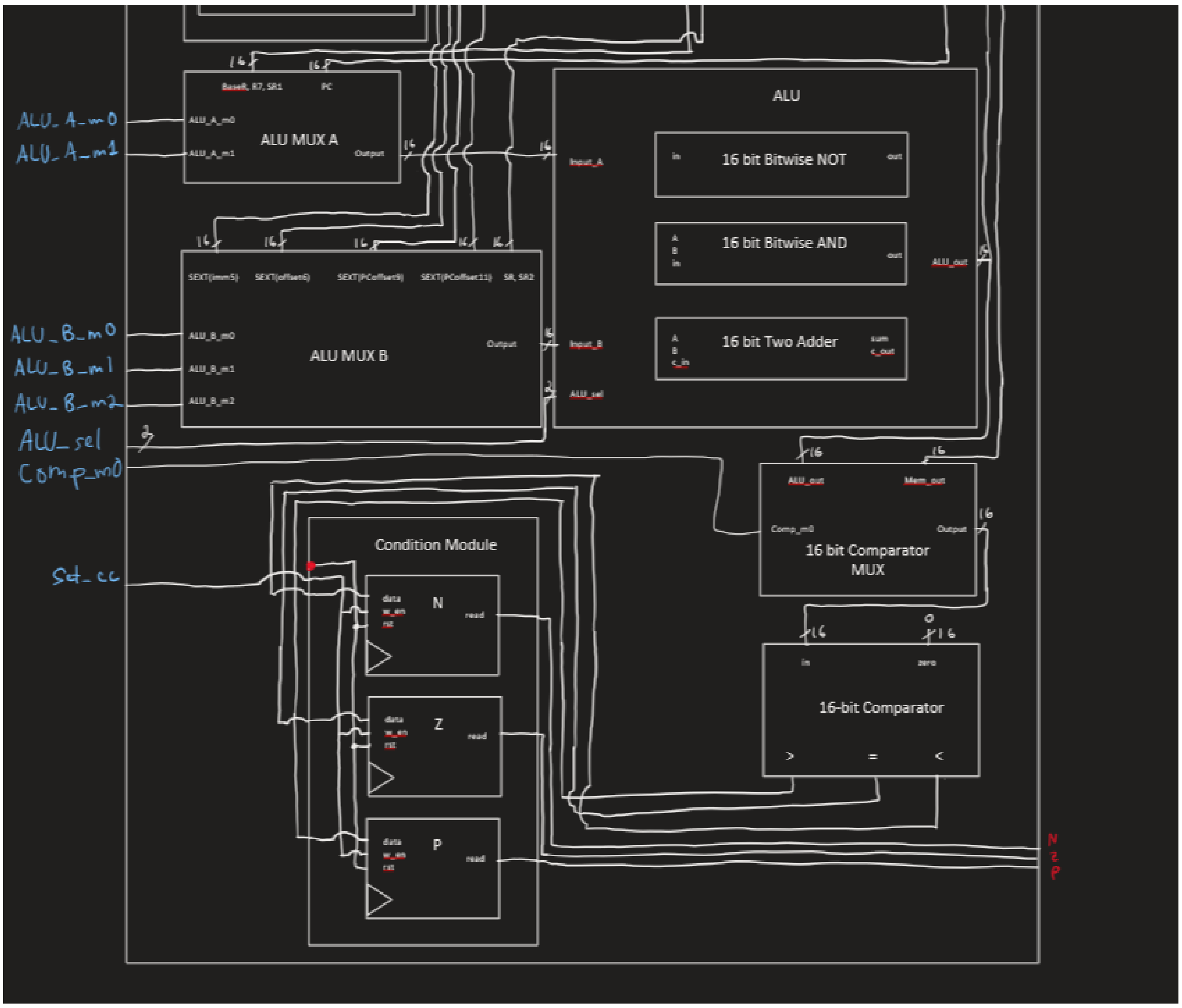


Figure 2.1.1: PUnC Datapath

2.2 Assembly Code

2.2.1 Program Description

The designed program is called "Bits". The program designed takes in one input - the maximum positive integer necessary to represent in a given data set. The program then has two outputs, which it stores in registers R5 and R3, as well as memory locations 26 and 27:

- The first output represents the minimum number of bits needed to represent all numbers in the data set (stored in R5 mem[26])
- The second output represents the maximum possible positive integer that can be represented with the given number of bits (stored in R3 mem[27])

2.2.2 Program in C

An initial version of the program was written in C, in a manner that was easily transferable to LC3 assembly. It is as follows:

```c
#include <stdio.h>

int main()
{
    int maxNumber = 245; //Change this number
    int maxPossible = 1;
    int temp1 = 0;
    int temp2 = 0;
    int exponent = 0;

    loop1:
    temp2 = temp1 - maxNumber;
    if(temp2 >= 0) goto endloop1; //if temp2 is zero or positive, branch to
   endloop1
        maxPossible += maxPossible;
        temp1 = maxPossible - 1;
        exponent++;
        goto loop1;
    endloop1:

    printf("The number of bits needed is: %d\n", exponent);
    printf("The maximum number in decimal is: %d\n", temp1);

    //Output:
    //The number of bits needed is: 8
    //The maximum number in decimal is: 255
}
```

Listing 1: Bits Program (C)

Figure 2.2.1: Bits Program in C

2.2.3 Program in Assembly

Then, the program was transferred to assembly. The assembly language program is as follows:

```
---- LOADS ----
LD R0, #21 //BaseR                              //0
LDR R1, R0, #1 //maxNumber                      //1
LDR R2, R0, #2 //maxPossible                    //2
LDR R3, R0, #3 //temp1                          //3
LDR R4, R0, #3 //temp2                          //4
LDR R5, R0, #3 //exponent                       //5
---- LOADS ----

---- LOOP 1 ----
NOT R6, R1 //temp2 = temp1 - maxNumber;         //6
ADD R6, R6, #1                                  //7
ADD R4, R3, R6                                  //8
BRp #9 //if(temp2 >= 0) goto endloop1;          //9
BRz #8                                          //10
ADD R2, R2, R2 //maxPossible += maxPossible;    //11
LDR R6, R0, #2 //temp1 = maxPossible - 1;       //12
NOT R6, R6                                      //13
ADD R6, R6, #1                                  //14
ADD R3, R2, R6                                  //15
LDR R6, R0, #2 //exponent++;                    //16
ADD R5, R5, R6                                  //17
BR #-13                                         //18
---- LOOP 1 ----

---- STORES ----
STR R3, R0, #5                                  //19
ST R5, #5                                       //20
HALT                                            //21
---- STORES ----

---- CONSTANTS ----
0016 //Offset for this location                 //22
00F5 //#245                                     //23
0001 //#1                                       //24
0000 //#0                                       //25
---- CONSTANTS ----

---- RESULTS ----
0000 //memory location for final exponent       //26
0000 //memory location for final temp1          //27
---- RESULTS ----
```

Listing 2: Bits Program (ASM)

Figure 2.2.2: Bits Program in LC3 Assembly

2.2.4 Program in Hex

Finally, the program was encoded in hexadecimal with the given assembler. The hex code is as follows:

```
2015   // LD R0, #21
6201   // LDR R1, R0, #1
6402   // LDR R2, R0, #2
6603   // LDR R3, R0, #3
6803   // LDR R4, R0, #3
6A03   // LDR R5, R0, #3
9C7F   // NOT R6, R1
1DA1   // ADD R6, R6, #1
18C6   // ADD R4, R3, R6
0209   // BRp #9
0408   // BRz #8
```

```
1482   // ADD R2, R2, R2
6C02   // LDR R6, R0, #2
9DBF   // NOT R6, R6
1DA1   // ADD R6, R6, #1
1686   // ADD R3, R2, R6
6C02   // LDR R6, R0, #2
1B46   // ADD R5, R5, R6
0FF3   // BR #-13
7605   // STR R3, R0, #5
3A05   // ST R5, #5
F000   // HALT
0016   // 0016
00F5   // 00F5
0001   // 0001
0000   // 0000
0000   // 0000
0000   // 0000
(Followed by 100 more rows of "0000")
```

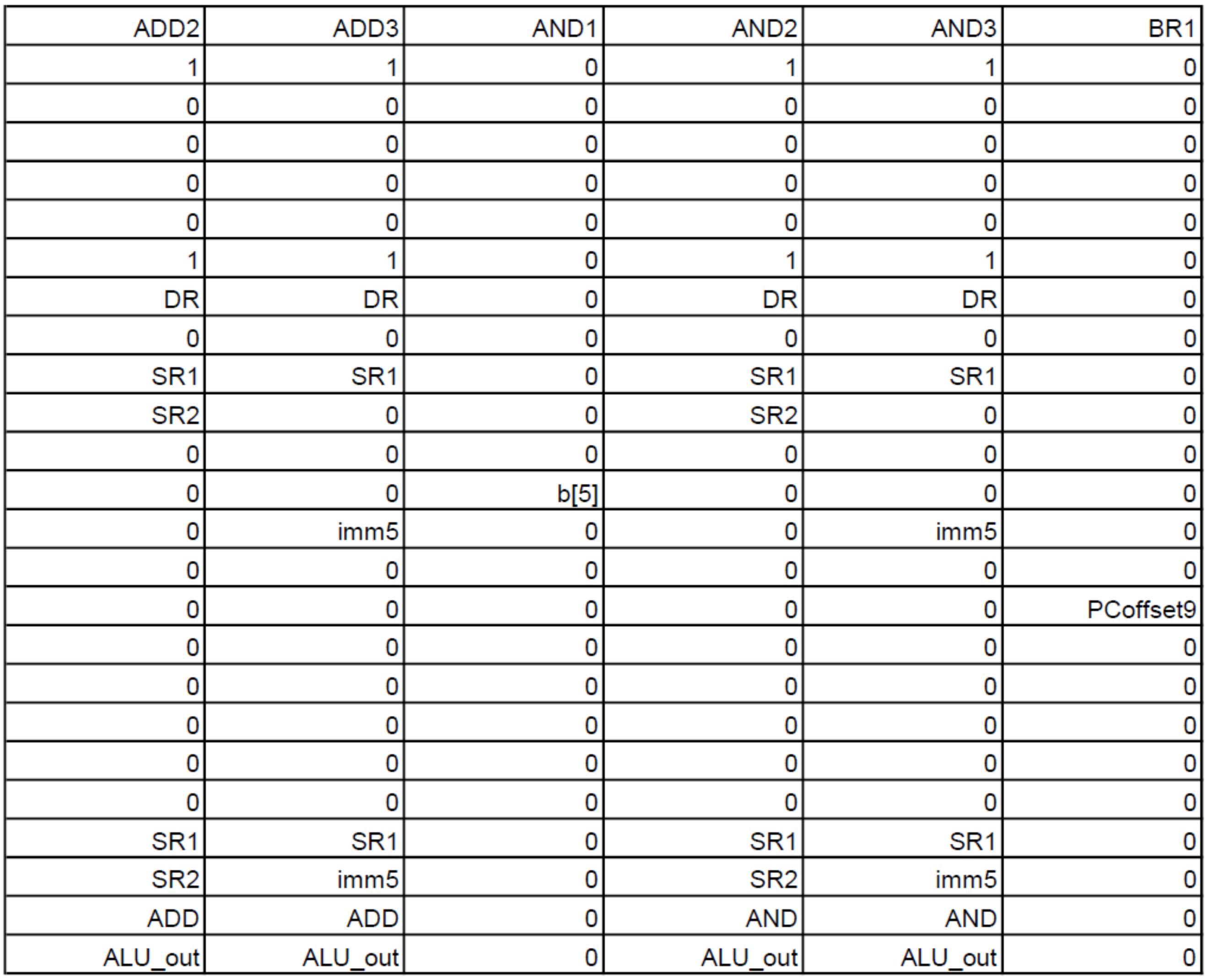
Listing 3: Bits Program (HEX)

Figure 2.2.3: Bits Program in Hex

2.3 Revised Control Signal Table

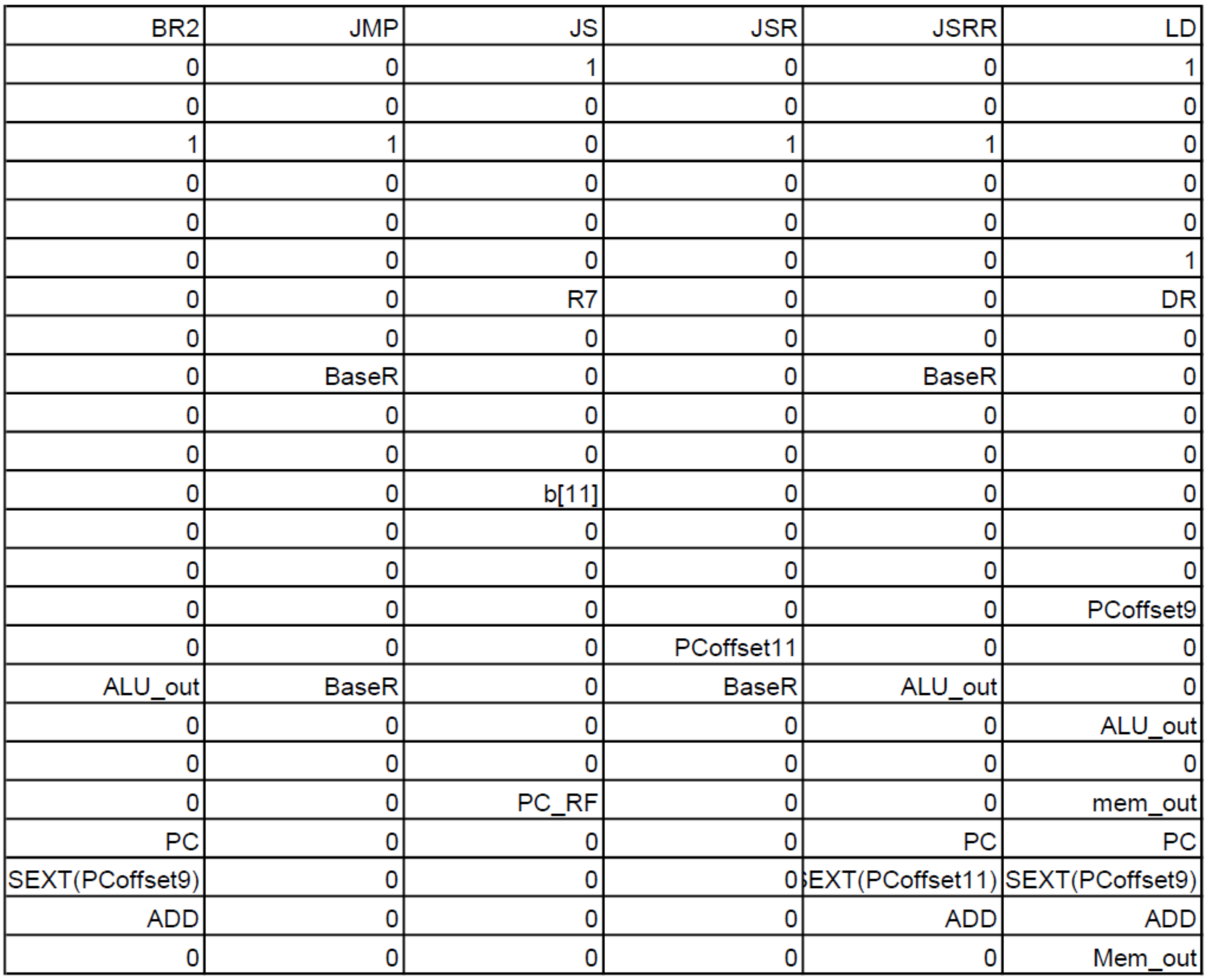
Control Signal	Fetch	Decode	ADD1
(rf_w_en) regfile_w_en	0	1	0
(mem_ld) mem_reg_ld	0	0	0
(PC_ld) PC_reg_ld	0	0	0
(PC_cnt) PC_reg_cnt	0	1	0
(IR_ld) IR_ld	1	0	0
(setcc) Set_cc	0	0	0
(dr_w_data) DR_addr	0	0	0
(mem_val_ld) mem_val_ld	0	0	0
(Rf1_addr_0) BaseR, R7, SR1 addr	0	0	0
(Rf2_addr_1) SR, SR2 addr	0	0	0
(instruction) instruction	instruction	0	0
(bit_in) Bit_in	0	0	b[5]
(imm5) imm5	0	0	0
(offset6) offset6	0	0	0
(PCoffset9) PCoffset9	0	0	0
(PCoffset11) PCoffset11	0	0	0
(PC_sel) PC_MUX Signal	0	0	0
(Mem_r_addr_sel) Mem_r_addr MUX Signal	0	0	0
(Mem_w_addr_sel) Mem_w_addr MUX Signal	0	0	0
(rf_w_data_sel) Regfile_w_data MUX Signal	0	0	0
(ALU_A_sel) ALU_A MUX Output Signal	0	0	0
(ALU_B_sel) ALU_B MUX Output Signal	0	0	0
(ALU_op) ALU_sel	0	0	0
(comp_sel) Comp_m0	0	0	0

Figure 2.3.1: Fetch, Decode, ADD1



ADD2	ADD3	AND1	AND2	AND3	BR1
1	1	0	1	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
1	1	0	1	1	0
DR	DR	0	DR	DR	0
0	0	0	0	0	0
SR1	SR1	0	SR1	SR1	0
SR2	0	0	SR2	0	0
0	0	0	0	0	0
0	0	b[5]	0	0	0
0	imm5	0	0	imm5	0
0	0	0	0	0	0
0	0	0	0	0	PCoffset9
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
SR1	SR1	0	SR1	SR1	0
SR2	imm5	0	SR2	imm5	0
ADD	ADD	0	AND	AND	0
ALU_out	ALU_out	0	ALU_out	ALU_out	0

Figure 2.3.2: ADD2, ADD3, AND1, AND2, AND3, BR1



BR2	JMP	JS	JSR	JSRR	LD
0	0	1	0	0	1
0	0	0	0	0	0
1	1	0	1	1	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	R7	0	0	DR
0	0	0	0	0	0
0	BaseR	0	0	BaseR	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	b[11]	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	PCoffset9
0	0	0	PCoffset11	0	0
ALU_out	BaseR	0	BaseR	ALU_out	0
0	0	0	0	0	ALU_out
0	0	0	0	0	0
0	0	PC_RF	0	0	mem_out
PC	0	0	0	PC	PC
SEXT(PCoffset9)	0	0	0	EXT(PCoffset11)	SEXT(PCoffset9)
ADD	0	0	0	ADD	ADD
0	0	0	0	0	Mem_out

Figure 2.3.3: BR2, JMP, JS, JSR, JSRR, LD

LDI1	LDI2	LDR	LEA	NOT	RET (unused)
0	1	1	1	1	0
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	0	0
0	0	0	0	0	0
0	1	1	1	1	0
0	DR	DR	DR	DR	0
1	0	0	0	0	0
0	0	BaseR	0	0	0
0	0	0	0	SR	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	offset6	0	0	0
PCoffset9	0	0	PCoffset9	0	0
0	0	0	0	0	0
0	0	0	0	0	R7
ALU_out	Mem_value	ALU_out	0	0	0
0	0	0	0	0	0
0	mem_out	mem_out	ALU_out	ALU_out	0
PC	0	BaseR	PC	0	0
SEXT(PCoffset9)	0	SEXT(offset6)	SEXT(PCoffset9)	SR	0
ADD	0	ADD	ADD	NOT	0
Mem_out	Mem_out	Mem_out	Mem_out	ALU_out	0

Figure 2.3.4: LDI1, LDI2, LDR, LEA, NOT, RET

ST	STI1	STI2	STR	HALT
0	0	0	0	0
1	0	1	1	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	0	BaseR	0
SR	0	SR	SR	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	offset6	0
PCoffset9	PCoffset9	0	0	0
0	0	0	0	0
0	0	0	0	0
0	ALU_out	0	0	0
ALU_out	0	Mem_value	ALU_out	0
0	0	0	0	0
PC	PC	0	BaseR	0
SEXT(PCoffset9)	SEXT(PCoffset9)	0	SEXT(offset6)	0
ADD	ADD	0	0	0
1	0	0	1	0

Figure 2.3.5: ST, STI1, STI2, STR, HALT

This paper represents my own work in accordance with University regulations.

Daniel Simone