In this file, we provide additional notes for the 10 functions in the project.c file.

1. **ALU function**

```
/*
  A: first operand for ALU
  B: second operand for ALU
  ALUControl: from value 0 (000) to 7 (111), where 0 represents +, 1 represents -, etc. (see
first table in Section 4 of the project-description for details).
  ALUresult: used to hold result, e.g., when ALUControl == 0, *ALUresult = A + B;
  Zero: *Zero = 1 if *ALUresult == 0 else *Zero = 0
*/
void ALU(unsigned A,unsigned B,char ALUControl,unsigned *ALUresult,char *Zero)
{

}
```

2. **instruction_fetch**

```
/*
Fetch the instruction addressed by PC from Mem, and write it to instruction.
Return 1 if a halt condition occurs; otherwise, return 0.

Notes:
1)  Specifically for instruction fetch, halt condition is a) if PC beyond range of 0x0000 to
    0xFFFF, OR 2) PC is not multiple of 4 (word-aligned).
2)  In spimcore.c, memory is defined as an unsigned array "Mem", and there is a macro
    provided to access Mem, i.e., MEM(addr). Therefore, you may use *instruction =
    MEM(PC) to read instruction memory.
*/

int instruction_fetch(unsigned PC,unsigned *Mem,unsigned *instruction)
```

3. **instruction_partition**

```
/*
unsigned op,        // instruction [31-26]
    r1,       // instruction [25-21]
    r2,       // instruction [20-16]
    r3,       // instruction [15-11]
    funct,  // instruction [5-0]
    offset, // instruction [15-0]
    jsec;    // instruction [25-0]
```

This function requires to get specific bits in “instruction” and assign it to *op, *r1, *r2, *r3, *func, *offset, *jsec. You may use bitwise operators in C (such as bitwise AND “&”, and right shift “>>” here. For example: *op = (instruction & 0xFC000000) >> 26;

Use this link as a reference: https://www.geeksforgeeks.org/extract-k-bits-given-position-number/
*/
void instruction_partition(unsigned instruction, unsigned *op, unsigned *r1,unsigned *r2, unsigned *r3, unsigned *funct, unsigned *offset, unsigned *jsec)

4. **instruction_decode**

/*
Update controls based on op. See below for details
*/
int instruction_decode(unsigned op,struct_controls *controls)

Below are the expected control signals for each type of instruction:

R type control signals
RegDst = 1
Jump = 0
Branch = 0
MemRead = 0
MemtoReg = 0
ALUOp = 7
MemWrite = 0
ALUSrc = 0
RegWrite = 1

Lw
RegDst = 0
Jump = 0
Branch = 0
MemRead = 1
MemtoReg = 1
ALUOp = 0
MemWrite = 0
ALUSrc = 1
RegWrite = 1

Sw
RegDst = DC
Jump = 0
Branch = 0
MemRead = 0

MemtoReg = DC
ALUOp = 0
MemWrite = 1
ALUSrc = 1
RegWrite = 0

Beq
RegDst = DC
Jump = 0
Branch = 1
MemRead = 0
MemtoReg = DC
ALUOp = 1
MemWrite = 0
ALUSrc = 0
RegWrite = 0

Slti
RegDst = 0
Jump = 0
Branch = 0
MemRead = 0
MemtoReg = 0
ALUOp = 2
MemWrite = 0
ALUSrc = 1
RegWrite = 1

Sltiu
RegDst = 0
Jump = 0
Branch = 0
MemRead = 0
MemtoReg = 0
ALUOp = 3
MemWrite = 0
ALUSrc = 1
RegWrite = 1

Addi
RegDst = 0
Jump = 0
Branch = 0
MemRead = 0
MemtoReg = 0
ALUOp = 0

MemWrite = 0
ALUSrc = 1
RegWrite = 1

Lui
RegDst = 0
Jump = 0
Branch = 0
MemRead = 0
MemtoReg = 0
ALUOp = 6
MemWrite = 0
ALUSrc = 1
RegWrite = 1

5. **read_register**

/*
Update data1 with Reg[r1], and update data2 with Reg[r2]
*/
void read_register(unsigned r1,unsigned r2,unsigned *Reg,unsigned *data1,unsigned *data2

6. **sign_extend**

/*
offset is a 32-bits number that has bits [31-0]

extended_value will contain value of "0b1111 1111 1111 1111 offset[15-0]" if offset[15] == 1, otherwise extended_value will be "0b0000 0000 0000 0000 offset[15-0]"
*/
void sign_extend(unsigned offset,unsigned *extended_value)

7. **ALU_operations**

/*
This function is expected to:
1) update ALUOp for R-type instructions; use slides page 111 of "Module5-MIPS" as a reference.
2) call ALU(…), note that the second parameter for ALU(…) can be data2 or extended_value based on ALUSrc.
*/
int ALU_operations(unsigned data1,unsigned data2,unsigned extended_value,unsigned funct,char ALUOp,char ALUSrc,unsigned *ALUresult,char *Zero) {

```
  if (ALUOp == 7) { // if this is a R-type instruction
```

```
        // Add instruction
        if (funct == 0b100000)
            ALUOp = 0;

        // implement for subtract, and, or, slt, sltu instructions

    ALU(data1, ???, ALUOp, ALUresult, Zero);
}
```

8. rw_memory

```
/*
Below is the pseudo-code for rw_memory function:

If MemRead != 0:
   If ALUresult is word-aligned:
      *memdata = data at memory address ALUresult
   else:
      return 1

 if MemWrite != 0:
    if ALUresult is word-aligned:
       data at memory address ALUresult = data2
    else:
       return 1
 return 0
*/
int rw_memory(unsigned ALUresult,unsigned data2,char MemWrite,char
MemRead,unsigned *memdata,unsigned *Mem)
```

9. write_register

/*
Here is what we expect in this function:

1) if RegWrite is 0, do nothing and return;
2) if RegWrite is not 0, update Reg; The register to be updated can be either r3 (rd) and r2 (rt), based on RegDst, the value used to update register can either be memdata or ALUresult, determined by MemtoReg. See page 51 and page 57 of "Module6-Processor" for details.

*/

```
void write_register(unsigned r2,unsigned r3,unsigned memdata,unsigned ALUresult,char
RegWrite,char RegDst,char MemtoReg,unsigned *Reg)
```

10. PC_update

```
/*
Here is the pseudocode for this function:

Update PC with PC+4

if (Branch && Zero):
   Update PC according to previous PC and extended_value // This is PC-relative addressing, Page 101 of “Module5-MIPS”

If (Jump):
   Update PC according to previous PC and jsec // This Pseudodirect addressing, Page 105 if “Module5-MIPS”
*/
void PC_update(unsigned jsec,unsigned extended_value,char Branch,char Jump,char Zero,unsigned *PC)
```