Project Help

Nazim Ashraf

Spimcore.c

```
void Step(void)
{
	/* fetch instruction from memory */
	Halt = instruction_fetch(PC,Mem,&instruction);

	if(!Halt)
	{
	/* partition the instruction */
	instruction_partition(instruction,&op,&r1,&r2,&r3,&funct,&offset,&jsec);

	/* instruction decode */
	Halt = instruction_decode(op,&controls);
	}

	if(!Halt)
	{
	/* read_register */
	read_register(r1,r2,Reg,&data1,&data2);

	/* sign_extend */
	sign_extend(offset,&extended_value);

	/* ALU */
	Halt = ALU_operations(data1,data2,extended_value,funct,controls.ALUOp,controls.ALUSrc,&ALUresult,&Zero);
	}

	if(!Halt)
	{
	/* read/write memory */
	Halt = rw_memory(ALUresult,data2,controls.MemWrite,controls.MemRead,&memdata,Mem);
	}

	if(!Halt)
	{
	/* write to register */
	write_register(r2,r3,memdata,ALUresult,controls.RegWrite,controls.RegDst,controls.MemtoReg,Reg);

	/* PC update */
	PC_update(jsec,extended_value,controls.Branch,controls.Jump,Zero,&PC);
	}
}
```

How to compile and run

Make an empty project
Add all 3 files to the project
From debug, add test file

Misconceptions

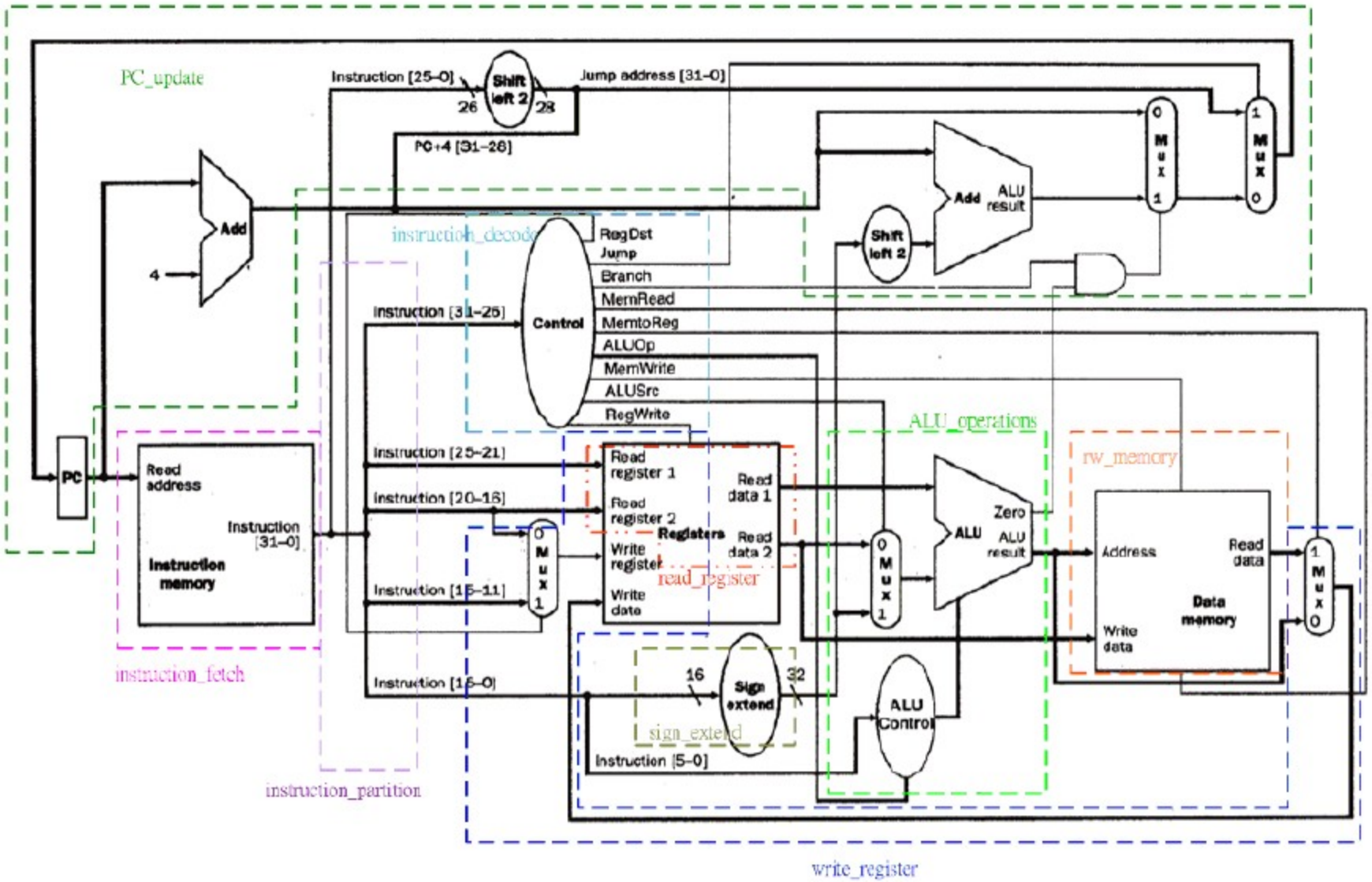
Do not have to make any changes to spimcore.c or spimcore.h
Do not have to worry about input/output
Do not have to convert between HEX and/or DEC

Controls

r	Dump registers contents
m	Dump memory contents (in Hexadecimal format)
s[n]	Step n instructions (simulate the next n instruction). If n is not typed, 1 is assumed
c	Continue (carry on the simulation until the program halts (with illegal instruction))
H	Check if the program has halted
d	ads1 ads2 Hexadecimal dump from address ads1 to ads2
I	Inquire memory size
P	Print the input file
g	Display all control signals
X, X, q, Q	Quit

Spimcore.c

```
void Step(void)
{
	/* fetch instruction from memory */
	Halt = instruction_fetch(PC,Mem,&instruction);

	if(!Halt)
	{
	/* partition the instruction */
	instruction_partition(instruction,&op,&r1,&r2,&r3,&funct,&offset,&jsec);

	/* instruction decode */
	Halt = instruction_decode(op,&controls);
	}

	if(!Halt)
	{
	/* read_register */
	read_register(r1,r2,Reg,&data1,&data2);

	/* sign_extend */
	sign_extend(offset,&extended_value);

	/* ALU */
	Halt = ALU_operations(data1,data2,extended_value,funct,controls.ALUOp,controls.ALUSrc,&ALUresult,&Zero);
	}

	if(!Halt)
	{
	/* read/write memory */
	Halt = rw_memory(ALUresult,data2,controls.MemWrite,controls.MemRead,&memdata,Mem);
	}

	if(!Halt)
	{
	/* write to register */
	write_register(r2,r3,memdata,ALUresult,controls.RegWrite,controls.RegDst,controls.MemtoReg,Reg);

	/* PC update */
	PC_update(jsec,extended_value,controls.Branch,controls.Jump,Zero,&PC);
	}
}
```


PC_update
Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
Add
4
M u x
0
1
Add
ALU result
Shift left 2
instruction_decode
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31–26]
Control
ALU_operations
rw_memory
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
read_register
ALU
Zero
ALU result
Address
Read data
Data memory
Write data
16
Sign extend
32
sign_extend
ALU Control
instruction_fetch
instruction_partition
write_register

How to start

Implement functions in order so that you can check each functions output…

IF

IP

ID

IF

int instruction_fetch(unsigned PC,unsigned *Mem,unsigned *instruction)
As mentioned before, Mem has already been populated, and PC should be the starting address
Check for word alignment
Use PC >> 2 to get the actual location

IP

```
void instruction_partition(unsigned instruction, unsigned *op, unsigned *r1,unsigned
*r2, unsigned *r3, unsigned *funct, unsigned *offset, unsigned *jsec)
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]
```

ID

```
int instruction_decode(unsigned op,struct_controls *controls)
typedef struct
{
        char RegDst;
        char Jump;
        char Branch;
        char MemRead;
        char MemtoReg;
        char ALUOp;
        char MemWrite;
        char ALUSrc;
        char RegWrite;
}struct_controls;
```

Read register

void read_register(unsigned r1,unsigned r2,unsigned *Reg,unsigned *data1,unsigned *data2)

Sign Extend

void sign_extend(unsigned offset,unsigned *extended_value)
16th bit is the sign bit
When we partitioned we put all zeros in the first 16 bits…

ALU_operations

int ALU_operations(unsigned data1,unsigned data2,unsigned extended_value,unsigned funct,char ALUOp,char ALUSrc,unsigned *ALUresult,char *Zero)

Set parameters for A, B, and ALUControl

If R-type instruction, look at funct

In the end, call

rw_memory

int rw_memory(unsigned ALUresult,unsigned data2,char MemWrite,char MemRead,unsigned *memdata,unsigned *Mem)
If MemWrite = 1, write into memory…
If MemRead = 1, read from memory

write_register

void write_register(unsigned r2,unsigned r3,unsigned memdata,unsigned ALUresult,char RegWrite,char RegDst,char MemtoReg,unsigned *Reg)
If RegWrite == 1, and MemtoReg ==1, then data coming from memory…
If RegWrite == 1, and MemtoReg ==0, then data coming from ALU_result

PC_update

void PC_update(unsigned jsec,unsigned extended_value,char Branch,char Jump,char Zero,unsigned *PC)

PC = PC + 4;

Take care of Branch and Jump

Zero – Branch taken or not

Jump: Left shift bits of jsec by 2 and use upper 4 bits of PC

Questions