Design for the Control Unit

The instruction will be fetched by using a Instruction fetch module and this instruction will be fed to a Instruction Decoder module. Which will decode the instruction using the following encoding:

Instruction [31:0] :

Opcode = instruction[31:26]

SourceReg1=instruction[25:21] for all I, R and B instructions

SourceReg2=instruction[20:16] for I and R

SourceReg3=instruction[15:11] for R instructions

Extended\_opcode=instructions[10:0] for R instructions

Immediate=instructions[15:0] for I type instructions

Offset=instructions[20:0] for Branch (B) type instructions

The instruction decoder will send the corresponding opcode and extended opcode to the control unit.

The control unit will have the following control signals as output:

1. ALU\_op (4 bit signal for ALU operation = Extended\_opcode[3:0])
2. reg\_dest (destination for R type instructions)
3. reg\_write (register write signal)
4. mem\_write (memory write signal, for Store operation)
5. mem\_read (memory read instruction for Load operation)
6. ALU\_src (decides between : 0 =register, 1=immediate for alu input)
7. branch (indicates if it is a branch operation)
8. memToReg (decides between SW and ALU output)

Truth Table for control signals:

For opcode 000000: ALU R type instructions

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction | Alu\_op | reg\_dest | reg\_write | mem\_write | Mem-read | ALU\_src | branch | memToReg |
| ADD | 0000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| SUB | 0001 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| AND | 0010 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| OR | 0011 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| XOR | 0100 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| NOT | 0101 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| SLA | 0110 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| SRA | 0111 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| SRL | 1000 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

\*reg\_dest denotes the presence of a 3rd register in the case of R type encoding

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction | Alu\_op | reg\_dest | reg\_write | mem\_write | Mem-read | ALU\_src | branch | memToReg |
| ADDI | 0000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| SUBI | 0001 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| ANDI | 0010 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| ORI | 0011 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| XORI | 0100 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| SLAI | 0110 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| SRAI | 0111 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| SRLI | 1000 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

Truth Table for Load and Store operations:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction | Alu\_op | reg\_dest | reg\_write | mem\_write | Mem-read | ALU\_src | branch | memToReg |
| LD | 0000 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| LDSP | 0000 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| ST | 0000 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| STSP | 0000 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

In the case of LD and ST instructions the offset value is first sign extended to 32 bits then added to the register value and then multiplied by 4 after which the mem\_read signal is sent for the particular memory address calculated.

Truth table for Branch type instructions:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction | Alu\_op | reg\_dest | reg\_write | mem\_write | Mem-read | ALU\_src | branch | memToReg |
| BR | XXXX | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| BMI | XXXX | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| BPL | XXXX | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| BZ | XXXX | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

If the instruction is a branch type instruction, then the branch control signal is set to 1, the ALU will take as input a register and produce signals sign, carry and zero, these signals will be sent to a Branch Control module that will consider the cases of BMI, BPL and BZ and depending on the condition defined by the opcode will set the conditional flag as 1 if the condition is met or else set the condition to 0, then the AND of Branch and conditional flag will be fed to a mux to select between PC + 4 and PC + 21-bit offset (The conditional flag will be set to 1 in case of BR instruction)

PC <= PC + imm if (Branch & conditional\_flag)

else

PC <= PC + 4

\*\* Some modifications are required in the ALU from previous week, like adding new output flags

Stack instructions:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction | Alu\_op | reg\_dest | reg\_write | mem\_write | Mem-read | ALU\_src | branch | memToReg |
| PUSH | 1001 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| POP | 0001 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| CALL | XXXX | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RET | XXXX | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

For PUSH operation we first increment SP to SP+4, by adding a new ALU operation encoded by 1001 which increments the R1 by 4 then set it to output, then we add Rin to R0 (0) that is the ALU\_out, then we send a mem\_write signal to write it into the memory of the stack

For POP operation we simply decrement the value of the Stack pointer by 4, this can be done by Subtracting 4 from SP that is

SP=SP-4, use ALU sub function in the ALU module with immediate value as 4;

For CALL function we do the following:

SP<=SP-4, Mem[SP]<=PC+4, PC<=PC+imm

For RET function we do:

PC=Mem[PC], SP<=SP+4

Move Instructions:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| instruction | Alu\_op | reg\_dest | reg\_write | mem\_write | Mem-read | ALU\_src | branch | memToReg |
| MOVE | 1010 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

\*ALU control signal 1010 returns A itself or in other words returns the First register value